DOI 10.31891/2307-5732-2025-347-23
УДК 621.395.721.5:004.77

ПАРХОМЕЙ ІГОР
Київський національний університет імені Тараса Шевченка
https://orcid.org/0000-0002-9510-7657
e-mail: i_parhomey@ukr.net
БОЙКО ЮЛІЙ
Хмельницький національний університет
https://orcid.org/0000-0003-0603-7827
e-mail: boiko_julius@ukr.net
ЛЕМЕШКО В'ЯЧЕСЛАВ
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
https://orcid.org/0009-0008-1495-715X
e-mail: slava.lemeshko@gmail.com

МАТЕМАТИЧНА МОДЕЛЬ ТА АДАПТИВНЕ УПРАВЛІННЯ АЛГОРИТМОМ ОБСЛУГОВУВАННЯ ДАНИХ ЖУРНАЛУ В УМОВАХ ЗМІННОГО СЕРВЕРНОГО НАВАНТАЖЕННЯ

У роботі досліджується проблема логування станів виконання бізнес-процесів та очищення застарілих логів у реляційній системі управління базами даних (СУБД). Запропоновано математичну модель і алгоритм динамічного управління процесом видалення/архівації даних на основі адаптивного порогу обробки (Nthreshold) та параметрів планування (Tnext_operation). Алгоритм реагує на зміни навантаження на сервер, коригуючи кількість записів для видалення та частоту запуску операцій. У якості головного критерію ефективності розглянуто баланс між обсягом оброблених даних, часом виконання операції та впливом на продуктивність системи. Експерименти показують, що внаслідок самоадаптації під час зниженої активності системи вдається збільшити кількість оброблених записів, а в періоди пікового навантаження алгоритм зменшує негативний вплив на продуктивність. Порівняння з відомими підходами свідчить про актуальність та перспективність застосування адаптивних методів керування фоновими завданнями (зокрема, очищення логів), що зумовлено необхідністю раціонального використання обчислювальних ресурсів і дотримання вимог щодо збереження та доступності даних.

Ключові слова: Адаптивний алгоритм, логування, системи управління базами даних, продуктивність, SQL.

PARKHOMEY IGOR
Taras Shevchenko National University of Kyiv
BOIKO JULIY
Khmelnytskyi National University
LEMESHKO VIACHESLAV
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

MATHEMATICAL MODEL AND ADAPTIVE CONTROL OF A DATA LOG MANAGEMENT ALGORITHM UNDER VARIABLE SERVER LOAD CONDITIONS

The article addresses the problem of optimizing the logging of business process execution states and managing outdated logs in a relational database management system (RDBMS). A mathematical model and an algorithm for dynamic log management are proposed, based on an adaptive processing threshold (Nthreshold) and scheduling parameters (Tnext_operation). The primary advantage of the approach lies in the algorithm's ability to adapt to server workload fluctuations by increasing the number of processed records during periods of low system activity and minimizing performance impact during peak loads. The algorithm adjusts the number of records to be deleted and the frequency of operations in response to changes in server load. The key efficiency criterion is the balance between the volume of processed data, operation execution time, and system performance impact. Experimental results demonstrate the algorithm's effectiveness in achieving a balance between system performance and the amount of processed data. The experiments show that, due to self-adaptation, the algorithm increases the number of processed records during periods of low system activity, while reducing its impact on performance during peak loads. The proposed methodology is promising for applications in business process automation, particularly in CRM systems (Customer Relationship Management), ERP systems (Enterprise Resource Planning), and e-commerce platforms, where stable server performance is critical. The use of relational databases combined with the proposed algorithm reduces implementation and training costs while maintaining high system flexibility and scalability. Comparison with well-known approaches highlights the relevance and prospects of applying adaptive methods for managing background tasks (particularly log cleaning), driven by the need for the efficient utilization of computational resources and compliance with data retention and availability requirements. Future research directions include extending the algorithm to support multi-user systems and integrating it with predictive analytics tools.

Keywords: Adaptive algorithm, logging, database management system, performance, structured query language.

Постановка проблеми

Логування стану виконання бізнес-процесів є операцією, що генерує великий обсяг даних [1]. З часом ці дані стають застарілими. Тому після певного періоду дані логування або архівуються, або видаляються зі сховища [2]. Оскільки операції архівації та видалення даних споживають ресурси віртуального сервера й виконуються у системі, яка продовжує генерувати лог-файли та обробляє запити для формування звітів на їх основі, питання мінімізації впливу на продуктивність платформи для виконання бізнес-сервісів є актуальним. Навіть за умови розгортання окремої системи логування на віртуальному сервері, проблема налаштування фонової задачі очищення даних логування залишається важливою [3].

Оптимізація процесу логування станів виконання бізнес-процесів та управління застарілими

логами в реляційній системі управління базами даних (СУБД) є важливою проблемою для забезпечення стабільної продуктивності інформаційних систем. В цьому випадку, основною метою оптимізації процесу логування є розробка математичної моделі та алгоритму динамічного керування очищенням логів, що базується на адаптивному порозі обробки (*Nthreshold*) та параметрах планування (*Tnext_operation*). Запропоноване рішення повинно забезпечувати адаптивність до змін навантаження на сервер, оптимізуючи кількість оброблюваних записів під час зниженої активності системи та мінімізуючи вплив на продуктивність у пікові періоди. Важливим аспектом дослідження є баланс між ефективністю обробки даних і продуктивністю системи. Додатково, дослідження повинно бути спрямоване на оцінку впливу запропонованого підходу на зниження витрат впровадження та навчання персоналу, зберігаючи гнучкість і масштабованість системи. Акцентуємо, що результати дослідження повинні демонструвати практичну цінність алгоритму для таких сфер, як CRM (Customer Relationship Management), ERP-системи (Enterprise Resource Planning) та платформ електронної комерції де необхідно забезпечити стабільну продуктивність серверів. Отже, мета роботи полягає в розробці математичної моделі та алгоритму динамічного керування очищенням логів у реляційних СУБД, що базується на адаптивному порозі обробки та параметрах планування. Алгоритм має забезпечити оптимізацію логування станів виконання бізнес-процесів та ефективне управління застарілими логами, зберігаючи баланс між продуктивністю системи та обсягом оброблюваних даних.

Аналіз останніх джерел

Як обговорювалось у роботах [1-3] екземпляри об’єктів, які спостерігаються являють собою ієрархічну структуру. Кожен екземпляр об’єкту у процесі свого існування переходить між певними стадіями. Шлях екземпляру від початкової стадії до кінцевої є його життєвим циклом. Дані логування, які згенеровані протягом його життєву циклу мають зберігатися певний період часу [4]. Застарілими даними слід вважати дані логування, які відносяться до екземпляру об’єкту, який знаходиться у завершеному стані, а також в яких сплив термін актуальності з моменту переходу екземпляру об’єкту у кінцеву стадію [5]. Поки об’єкт не перейшов у кінцеву стадію і дата переходу у кінцеву стадію не відповідає терміну старіння даних, то всі дані пов’язані із цим екземпляром об’єкта мають зберігатися, незалежно від дати їх логування. Це потрібно, щоб у будь-який момент часу було можливо сформувати звіти про стан конкретного екземпляру об’єкту або сформувати інші агреговані звіти про стани спостережуваних екземплярів об’єктів. Згідно з вимогами та природою даних, які зберігаються, запропоновано використовувати реляційну СУБД як сховище даних логування [6]. Створено реляційні таблиці для зберігання станів екземплярів об’єктів. Зв'язок між даними організовано з допомогою зовнішніх ключів. Реляційна СУБД дозволяє використовувати існуючі, стандартні фрейм ворки, наприклад, Entity frame work [7], які дозволяють організовувати CRUD-операцій для управління даними та формування звітів статистик необхідних для аналізу якості автоматизації бізнес-сервісів [8-10]. “Стандартні” реляційні СУБД пропонують різноманітні інструменти для налагоджування та аналізу продуктивності *SQL*-запитів, що є вагомим фактором використання таких сховищ даних [11, 12]. А також не потребують додаткових видатків на навчання для співробітників, ніж це було б необхідним у випадку із іншими типами сховищ даних. В даній роботі розглядається параметрична оптимізація підсистеми логування, а саме мінімізація впливу процесу очищення ієрархічних даних логування на навантаження віртуального серверу. Отже, як і у роботах [13], [14], пропонується алгоритми без радикальної перебудови СУБД - натомість використано традиційні засоби (індекси, зовнішні ключі, *SQL*-запити, Entity Framework тощо) та скрипти «очищення».

Опис алгоритму обслуговування даних журналу

Запропонований алгоритм обслуговування даних журналу на основі наведеної блок-схеми (рис. 1) працює наступним чином.

1. Алгоритм починається з ініціалізації основних параметрів (табл. 1). Метою даного кроку є підготовка параметрів для роботи алгоритму.

Таблиця 1

Опис параметрів налаштування задачі обслуговування даних журналу

Назва	Значення	Заголовок	Опис
TaskTimeout	1	Максимальний час виконання задачі обслуговування даних журналу (хвилин)	Якщо деякі дані не були оброблені протягом зазначеного часу, обробка даних продовжиться під час наступного запуску операції обслуговування.
TaskFrequencyMinutes	5	Частота виконання задачі для обслуговування даних журналу (хвилин)	
LogExpirationPeriod	30	Термін актуальності даних журналу (днів)	Після закінчення зазначеного періоду верхнерівневі дані, які переведені в кінцеву стадію автоматично видаляються з журналу

Продовження таблиці 1

Назва	Значення	Заголовок	Опис
RootDataDeletionThreshold	100	Максимальна кількість верхньорівневих елементів ієрархії, які можуть бути видалені за одну операцію	Визначає правила видалення основних даних (даних верхнього рівня). Наприклад, це може бути мінімальна кількість дочірніх записів, яка має бути видалена перед видаленням запису верхнього рівня. вузлів структури.
SettingsExpirationTime		Час життя налаштувань	Цей параметр забезпечує гнучкість і адаптивність системи, дозволяючи уникати використання застарілих налаштувань. Параметр визначає час, після якого поточні налаштування втрачають свою актуальність і повинні бути переосмислені або оновлені. Значення задається у вигляді конкретної дати та часу.
Tidle	120	Технічне вікно обслуговування (хвилин)	Проміжок часу в межах якого можливо максимально використовувати ресурси віртуального серверу

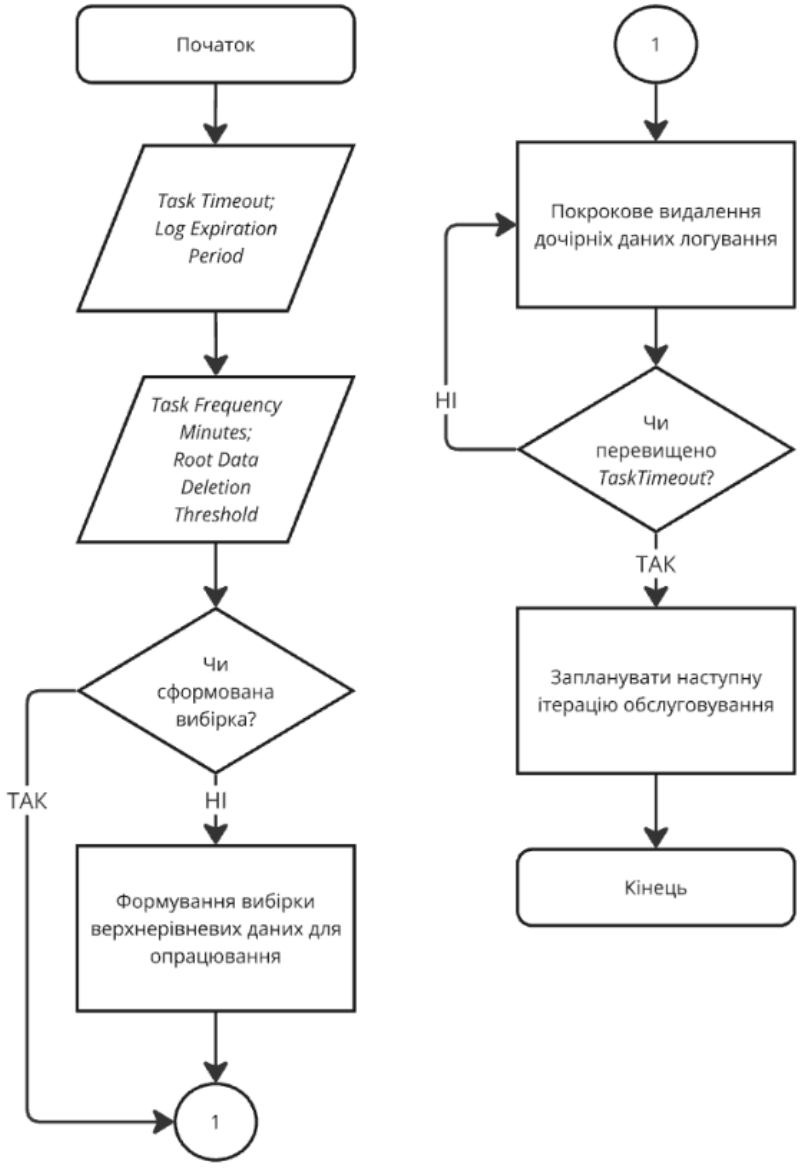

Рис.1. Алгоритм виконання задачі обслуговування даних журналу

2. Далі відбувається перевірка, чи існує вибірка верхньорівневих даних для обробки. Це потрібно для того, щоб зменшити навантаження на систему, працюючи тільки з релевантними даними. Якщо вибірка не сформована: виконується її формування (крок 3). Якщо вибірка готова, то алгоритм переходить до покрокового видалення дочірніх даних. Метою кроку є забезпечення, щоб алгоритм працював лише з необхідними даними, оптимізуючи ресурсне навантаження.

3. Формується список записів верхнього рівня, які відповідають параметру *LogExpirationPeriod*. Наприклад, вибираються дані журналу, створені більше 30 днів тому (якщо *LogExpirationPeriod* = 30). Виконується *SQL*-запит або логіка відбору на основі СУБД. Обираються лише ті записи, які задовольняють обом умовам (*LogExpirationPeriod* та верхнерівневі записи). Метою кроку є формування списку даних для видалення.

4. На цьому етапі алгоритм починає видаляти дочірні дані записів журналу з урахуванням *RootDataDeletionThreshold*, який визначає кількість записів, які обробляються (видаляються) за один цикл. Значення *RootDataDeletionThreshold* змінюється динамічно залежно від швидкості видалення даних. Якщо видалення виконується швидко, значення параметра може бути збільшене та навантаження на СУБД. Якщо система перевантажена, значення *RootDataDeletionThreshold* зменшується, щоб мінімізувати вплив на базу даних. Алгоритм починає видалення дочірніх даних для записів, відібраних на кроці 3. Початкове значення *RootDataDeletionThreshold* задається, наприклад, 100 записів. Видалення виконується покроково, обробляючи за раз не більше *RootDataDeletionThreshold* дочірніх записів. Після кожного кроку перевіряється залишок часу від *TaskTimeout*. Аналіз швидкості видалення та адаптація *RootDataDeletionThreshold*: якщо видалення даних завершується швидше, ніж очікувалося (залишається значний час у межах *TaskTimeout*), *RootDataDeletionThreshold* збільшується, наприклад, на 20–30%. Якщо видалення даних сповільнюється через перевантаження системи, *RootDataDeletionThreshold* зменшується (наприклад, з 100 до 50 записів за ітерацію). Якщо всі дочірні записи для вибраного запису верхнього рівня видалено, алгоритм переходить до наступного запису. Якщо перевищено *TaskTimeout* до завершення обробки, робота припиняється і планується наступна ітерація. Метою адаптації *RootDataDeletionThreshold* є збереженням балансу між швидкістю обробки та стабільністю системи та мінімізувати вплив на продуктивність СУБД під час пікових навантажень.

5. Якщо ітерація була зупинена через перевищення *TaskTimeout*, алгоритм планує запуск наступної ітерації через інтервал *TaskFrequencyMinutes*. Наприклад, алгоритм зупинився о 14:00, то наступна ітерація буде запущена о 14:10, якщо значення *TaskFrequencyMinutes* = 10. Якщо вибірка порожня, то запуск алгоритму реєструється на початок наступної доби адже неактуальність даних визначається в днях. Даний крок гарантує безперервну обробку даних, уникаючи перевантаження.

6. Завершуються усі завдання для даної ітерації, не залишаючи зайвих процесів та тимчасових даних.

Це дозволяє підтримувати стабільність і ефективність роботи навіть при великому обсязі даних.

Вибір критерію ефективності та опис математичної моделі процесу обслуговування даних

Критерій ефективності має враховувати не тільки обсяг оброблених даних та вплив на продуктивність системи, але й ефективність використання часу, протягом якого можна використовувати всі ресурси сервера, а також оптимальність планування наступного запуску операції.

Формула критерію ефективності:

$$E = Nprocessed / (Toperation * Cimpact + Tidle), \quad (1)$$

де E - критерій ефективності операції; *Nprocessed* - кількість записів, оброблених за період обслуговування; *Toperation* - реальний час виконання операції (у хвилинах); *Cimpact* - коефіцієнт впливу на продуктивність системи; *Tidle* - час, протягом якого сервер залишається бездіяльним (наприклад, між операціями через неоптимальне планування).

Тоді критерій ефективності максимізується, коли *Nprocessed* максимальний (оброблено більше даних). *Toperation* та *Cimpact* мінімізовані. *Tidle* скорочено за рахунок оптимального планування запуску операції.

Запропонований критерій (1) корелює з ідеями вартості/корисності з [10] та [11], де критерії або мінімізують вартість зберігання, або максимізують пропускну здатність при заданому рівні ресурсів.

Також роботи [5], [9] і [14] приділяють увагу тому, що фонові задачі варто запускати під час мінімального завантаження або “вікон” (*maintenance win*dow). Якщо сервер має 30-хвилинне вікно з невисоким навантаженням, алгоритм різко збільшує *Nthreshold*, аби обробити максимум застарілих логів.

Критерій ефективності можна сформулювати іншими словами, наприклад, обробити усі дані за добу при цьому не навантажуючи систему.

До параметрів моделі можна виділити наступні групи.

1. Стан роботи алгоритму: *Nthreshold* - поріг кількості записів для обробки в одній ітерації; *Ttimeout* - максимальний час роботи операції (*TaskTimeout*); *Tprocess* - середній час обробки одного запису; *Lload* - поточне навантаження на сервер.

2. Часові параметри: *Tfull_utilization* - час (у хвилинах), протягом якого операція може використовувати всі ресурси сервера; *Tnext_operation* - час до наступного запуску операції

(*TaskFrequencyMinutes*).

Розглянемо питання динамічної адаптації параметрів.

У більшості аналогічних робіт фігурує ідея динамічної зміни ключових змінних: поріг обробки, обсяг даних, розмір пакетів тощо залежно від поточного або прогнозованого навантаження. У даній роботі ми це реалізуємо через *Nthreshold* та α, *β* (2), а також через *Tnext_operation* (3). Налаштування параметру *Nthreshold*, адаптація порогу видалення виконується з урахуванням наступного виразу:

$$Nthreshold\_new = min(Nthreshold\_old*\alpha*\beta, Nmax), \quad (2)$$

де: $\alpha = (Ttimeout - Tused)/Ttimeout$ - коефіцієнт, що враховує залишок часу; $\beta = 1/Lload$ - коефіцієнт, що враховує поточне навантаження на сервер; *Nmax* - максимальне значення *Nthreshold*, визначене конфігурацією.

Оптимізація параметру *Tnext_operation*, час планування наступного запуску залежить від поточного стану системи:

$$Tnext\_operation = max(Toperation + Tbuffer, Tfull\_utilization - Toperation), \quad (3)$$

де: *Tbuffer* - буферний час для стабілізації системи між операціями; *Tfull_utilization* - період, коли сервер може працювати з повним навантаженням.

Метою даної моделі є задача максимізації критерію ефективності *E*, враховуючи часові та ресурсні обмеження у відповідності до виразу:

$$\max Nthreshold, Tnext\_operation\ (E = Nprocessed / (Toperation*Cimpact + Tidle)) \quad (4)$$

Результати експериментальних досліджень

Випробування роботи системи проводилося в інфраструктурі із такими характеристиками: Web server: 4 cores, 32 GB RAM; *SQL* server: 4 cores, 32 GB RAM (1 databases).

В ході випробовування було визначено, що робота моделі не впливає на такі характеристики як Web server CPU usage та Web server memory usage. Тому вони не будуть наведені в даній роботі.

Такі роботи як [1], [10] наголошують на тому, що операції видалення чи архівації можуть спричиняти зростання *I/O* і блокування транзакцій. Тому для аналізу моделі пропонується розглянути наступні характеристики: *sql IOPS*, *sql processes blocked*, *sql disk queue*, що підтверджує важливість підходу.

Рисунок 2 відображає графік вплив алгоритму самоналаштування на підсистему вводу/виводу (IOPS) для журнала даних логування. Його можна умовно розділити на три фази. Перша фаза відображає стан «звичайного режиму», де присутня висока нестабільність і стрибкоподібність показника IOPS, що свідчить про значне навантаження на підсистему вводу/виводу. Система працює без алгоритму оптимізації, що призводить до інтенсивного використання ресурсів. На другій фазі відбувається адаптація (другий інтервал).

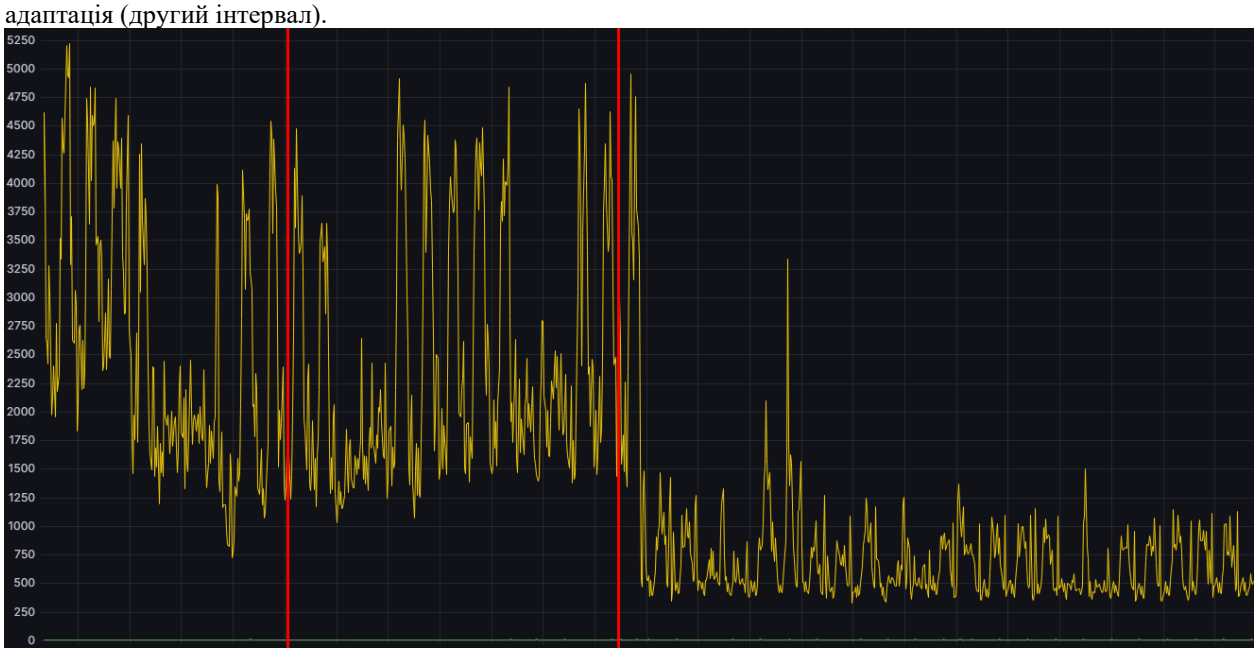

Рис.2. Показник *sql IOPS*

Спостерігається поступове зниження IOPS через впровадження алгоритму динамічного налаштування. Алгоритм регулює параметри, такі як поріг обробки записів (*Nthreshold*) і частоту запуску операцій (*Tnext_operation*). На третій фазі (третій інтервал) спостерігається робота системи в оптимізованому стані. Значне зниження та стабілізація IOPS на низькому рівні. Система знаходиться у стані ефективної роботи: зменшено вплив на підсистему вводу/виводу, забезпечено рівномірне використання ресурсів [15, 16].

Рисунок 3 демонструє залежність зміни кількості заблокованих *SQL*-процесів у часі, який розділений на три етапи.

Рис.3. Показник *sql processes blocked*

На першій фазі спостерігається нерівномірне та високе блокування процесів. Це свідчить про значні проблеми з конкуренцією ресурсів у системі. На другій фазі, фазі адаптації, відбувається то збільшення то зменшення кількості заблокованих процесів, що вказує на активізацію алгоритмів самоналаштування. Система намагається налаштуватися на ефективне розподілення ресурсів. На третій фазі спостерігається оптимізований стан: мінімальна кількість заблокованих процесів, стабільність та ефективна робота системи.

Рисунок 4 відображає графік зміни черги доступу до диска у системі *SQL* Server.

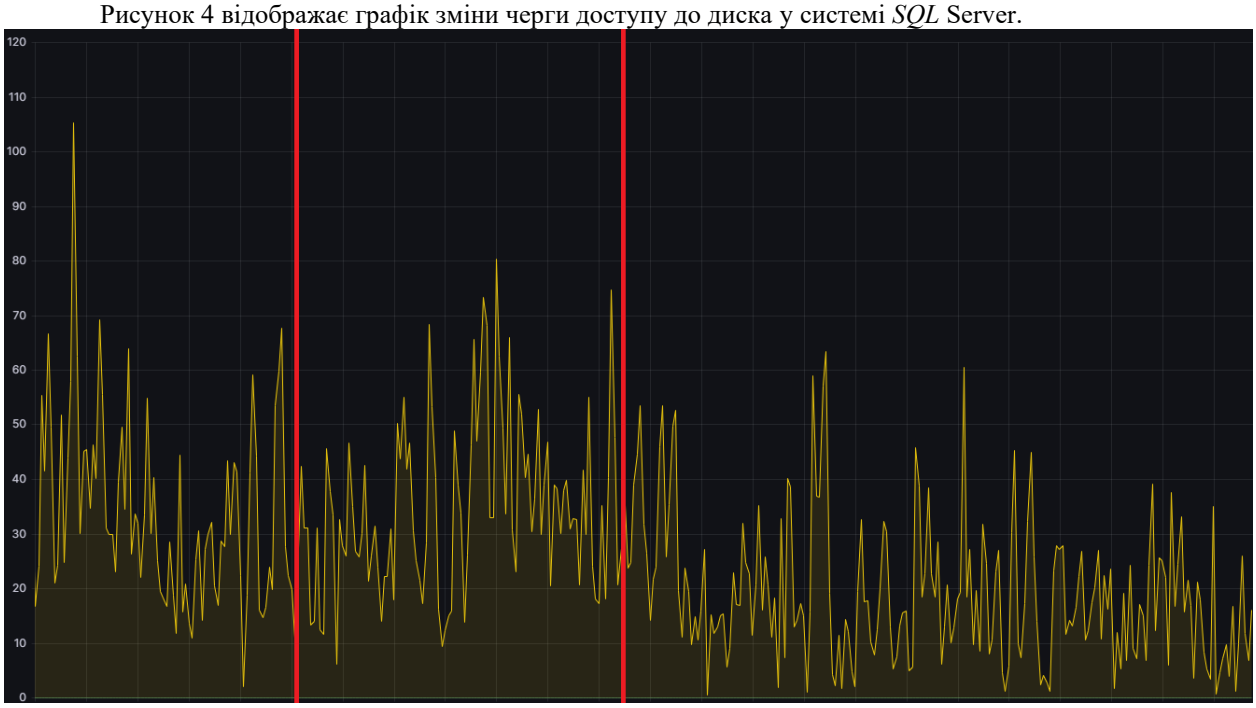

Рис.4. Показник *sql disk queue*

На першій фазі спостерігається високі коливання черги диска через нерівномірний розподіл завдань. Система працює без оптимізації доступу до дискових ресурсів. На другій фазі відбувається адаптація. Спостерігається поступове зменшення амплітуди коливань. Це свідчить про активізацію алгоритму, що покращує розподіл операцій на диску. На третій фазі черга диска стабілізується, ресурси використовуються ефективно. Мінімальне очікування в черзі, що свідчить про оптимальне налаштування доступу.

Висновки

Запропонована модель та алгоритм обслуговування даних журналу демонструють ефективність підходу динамічного управління параметрами очищення логів з урахуванням поточного навантаження на сервер. Дослідження підтверджує, що адаптивна зміна порогу обробки (*Nthreshold*) дає змогу збільшити обсяг видалених або архівованих записів у «вікна» низької зайнятості системи без погіршення продуктивності під час пікового навантаження. Гнучке планування операцій (*Tnext_operation*) уникає зайвих простоїв і забезпечує рівномірне розподілення ресурсу на процес очищення, завдяки чому зменшується кількість заблокованих процесів у СУБД та знижується навантаження на підсистему введення/виведення (IOPS).

Критерій ефективності на основі співвідношення між кількістю оброблених логів, витраченим часом та впливом на продуктивність дає змогу кількісно оцінити оптимальність обраного режиму роботи алгоритму. Порівняння з аналогічними підходами [1–14] у сфері керування даними та фонового обслуговування баз даних підтверджує, що запропоноване рішення органічно поєднує перевірені методи (планування фонового завдання, робота з ієрархічними логами) з адаптивними механізмами самоналаштування.

Передбачаються наступні подальші напрями досліджень:

- поглиблене використання методів машинного навчання для більш точної прогностики навантаження;
- розширення алгоритму очищення на мікросервісну або розподілену інфраструктуру;
- вдосконалення системи індексування реляційної СУБД з метою прискорення операцій вибірки та видалення логів.

Таким чином, адаптивне управління фоновим видаленням застарілих даних логування дає можливість ефективно утримувати високу продуктивність платформи й водночас дотримуватися вимог щодо зберігання та аналітики даних.

References

1. Le, T., Azevedo, L., Simões, P., & Nunes, M. (2021). Adaptive log management for large-scale distributed systems. *Journal of Grid Computing, 19*(1), 1–18.
2. He, P., Zhu, J., He, S., Li, J., & Lyu, M. R. (2018). Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing, 15*(6), 931–944.
3. Yao, K., Sayagh, M., Shang, W., & Hassan, A. E. (2022). Improving state-of-the-art compression techniques for log management tools. *IEEE Transactions on Software Engineering, 48*(8), 2748–2760.
4. Chen, X., Park, C., & Shin, K. G. (2018). Ensuring required disk I/O performance in virtualized clouds. *IEEE Transactions on Cloud Computing, 6*(1), 98–111.
5. Yassa, F., Boussaid, O., & Bentayeb, F. (2018). Log-based ETL for real-time data warehousing. *Journal of Big Data, 5*(1), 1–24.
6. Kvet, M., Papan, J., & Durneková, M. H. (2024). Treating temporal function references in relational database management system. *IEEE Access, 12*, 54518–54535.
7. Microsoft. (2024). Entity Framework documentation hub. Retrieved from https://learn.microsoft.com/en-us/ef/ (last access: 21.12.2024).
8. Theodoropoulos, G. K., & Bouzeghoub, A. (2020). Adaptive and self-tuning data cleansing in real-time analytics. *Future Generation Computer Systems, 110*, 940–955.
9. Zhu, S., Ji, C., & Li, T. (2019). A resource adaptive scheduling algorithm in clouds. *Journal of Parallel and Distributed Computing, 127*, 165–177.
10. Kang, D., Choi, S., & Chien, A. A. (2020). Empirical analysis of task scheduling in serverless computing platforms. In *Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (pp. 19–28).
11. Abadi, D. J., Maddox, R., & Samuel, D. (2020). Cost-optimized data retention for cloud data warehouses. *Proceedings of the VLDB Endowment, 13*(9), 1334–1346.
12. Chaudhuri, S., & Narasayya, V. (2021). Automated physical design tuning: Workload as a missing link. *ACM SIGMOD Record, 50*(3), 28–36.
13. Liu, L., Zhou, X., & Qiao, M. (2022). Adaptive parameter tuning for data cleaning operators in stream processing systems. *Journal of Systems and Software, 188*, 111278.
14. Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hugg, J., & Stolze, K. (2007). The end of an architectural era (it's time for a complete rewrite). *Proceedings of the VLDB Endowment, 12*(2), 1150–1160.
15. Пархомей, І., Бойко, Ю., & Лемешко, В. (2024). Алгоритм налаштування кількості потоків для виконання фонових задач. *Вимірювальні та обчислювальні прилади в технологічних процесах*, (4), 162–173.
16. Parkhomey, I., Boiko, J., Tsopa, N., Zeniv, I., & Eromenko, O. (2020). Assessment of quality indicators of the automatic control system influence of accident interference. *Telkomnika (Telecommunication Computing Electronics and Control), 18*(4), 2070–2079.