**УДК 004.9**
**DOI: https://doi.org/ 10.17721/3041-2323.2024.161-171**

**Ярослав ЖИЛЮК, асп.**
**ORCID ID: 0009-0008-9341-9164**
**e-mail: y.zhyliuk@gmail.com**
**Київський національний університет**
**імені Тараса Шевченка, Київ, Україна**

# ВІДМОВОСТІЙКА АРХІТЕКТУРА ПОТОКОВОГО ОБРОБЛЕННЯ ДЛЯ АНАЛІЗУ ДАНИХ У РЕАЛЬНОМУ ЧАСІ

***Розглянуто зростаючий попит на оброблення безперервних потоків даних із малою затримкою у сфері аналізу даних у реальному часі, досліджено існуючі підходи до відмовостійкості в розподілених системах оброблення потоків даних і запропоновано нову гібридну архітектуру асинхронної взаємної реплікації. Ця архітектура розроблена для розв'язання проблем, пов'язаних із затримкою та перебоями в мережі, і має мінімальний вплив на узгодженість даних і доступність на вузлах. Розглянуто механізми забезпечення стабільності передачі даних та архітектура системи для підвищення продуктивності систем оброблення розподілених потоків.***

**Ключові слова:** ***відмовостійкість, потокові дані, розподілені системи, реплікація даних, аналітика даних у реальному часі.***

## Вступ

Зростаючий попит на оброблення даних у режимі реального часу спонукає впроваджувати нові механізми реплікації, що критично важливо для різних галузей, включаючи фінансовий сектор, промисловість, телекомунікації та Інтернет речей. Потокове оброблення даних дозволяє своєчасно отримувати інформацію та оперативно реагувати на зміни в інформаційних системах різного призначення (Carbone, 2015).

Однак забезпечення відмовостійкості в таких системах за збереження швидкості їхньої роботи залишається критичною проблемою. За своєю природою розподілені інформаційні системи вразливі до збоїв, що можуть серйозно вплинути на надійність і ефективність аналітичних програм у реальному часі (Глоба, 2013). Надійність реплікації даних є ключовим аспектом, що забезпечує

безперервне та точне оброблення інформації для підтримки критичних процесів прийняття рішень (Білокон, 2024; Журавель, Думич, & Шпур, 2021).

**Результати**

Є різні підходи до розв'язання проблеми відмовостійкості в розподілених системах потокового оброблення даних, що базуються на принципах роботи розподілених систем, і консенсусних алгоритмів для пом'якшення впливу збоїв. Одна з класичних стратегій реплікації, відома як реплікація "лідер-послідовник", що забезпечує надійність розповсюдження через реплікацію даних між кількома брокерами розподіленої інформаційної системи (Kreps, Narkhede, & Rao, 2011). Лідер відповідає за приймання і оброблення записів, тоді як послідовники пасивно реплікують дані від лідера. У випадку збою лідера один із послідовників стає новим лідером, що визначається обраним алгоритмом консенсусу.

Стратегії гібридної реплікації поєднують синхронну й асинхронну реплікацію для балансування узгодженості та доступності в розподілених системах. У системі Spanner's TrueTime синхронна реплікація підтверджує оновлення даних лише після їхньої реплікації в кілька розподілених вузлів (Corbett, 2012), що забезпечує надійну узгодженість, але може викликати затримки та зменшити доступність, особливо, якщо репліки географічно розподілені. Асинхронна реплікація дозволяє локально підтверджувати оновлення перед розповсюдженням на інші вузли, покращуючи доступність і зменшуючи затримку, але потенційно призводить до розбіжностей у даних (DeCandia, 2007).

Попри спроби мінімізувати затримку за допомогою гібридних стратегій реплікації, необхідність підтримувати стійку узгодженість між розподіленими репліками все ще може призвести до затримок порівняно із системами, які надають пріоритет остаточній узгодженості або призначені для операцій із меншою затримкою.

Продуктивність і надійність таких систем значною мірою залежать від якості та доступності мережі, що з'єднує центри оброблення даних, а мережеві збої та переривання можуть потенційно вплинути на здатність підтримувати узгодженість і доступність реплік, що створює додаткові виклики для забезпечення відмовостійкості й ефективного оброблення даних у реальному часі.

Однією з головних проблем у розподілених потокових системах є досягнення балансу між узгодженістю даних і доступністю системи. Теорема CAP (Consistency, Availability, Partition Tolerance) стверджує, що розподілена система не може водночас гарантувати всі три властивості (Brewer, 2000).

У контексті систем оброблення даних у реальному часі це означає, що необхідно робити компроміси між узгодженістю та доступністю, особливо в умовах мережевих розподілів.

Асинхронна реплікація покращує доступність і зменшує затримки, оскільки дозволяє підтверджувати операції локально, однак це може призвести до розбіжностей у даних між вузлами, оскільки оновлення можуть не встигати синхронізуватися вчасно. У критично важливих системах, де узгодженість даних є пріоритетом, такі розбіжності можуть мати серйозні наслідки (Lakshman, & Malik, 2010).

Алгоритми консенсусу, такі як Raft або Paxos, забезпечують узгодженість у розподілених системах, але можуть бути складними для реалізації та призводити до значних накладних витрат (Lamport, 2001). Вони зазвичай вимагають багатьох раундів обміну повідомленнями, що збільшує затримки. Крім того, вони не завжди оптимізовані для систем із низькою затримкою, що є критичним для оброблення даних у реальному часі.

***Гібридна модель асинхронної реплікації.*** Розроблення гібридної архітектури асинхронної взаємної реплікації може розв'язати проблему затримок і мережевих переривань, мінімально впливаючи на узгодженість і доступність даних у вузлах.

З огляду на обмеження наявних підходів, виникає потреба в такій архітектурі, яка поєднувала б переваги синхронної та асинхронної реплікацій, мінімізуючи їхні недоліки (Lin, 2017). Ціль дослідження – представити та протестувати архітектуру оброблення потокових даних у реальному часі, що здатна забезпечити відносно низьку затримку у процесі збереження прийнятного рівня узгодженості та відмовостійкості.

Ключовим елементом запропонованого підходу є поняття "найближчого вузла", що визначається на основі часу затримки під час обміну повідомленнями між вузлами, що дозволяє організувати

реплікацію таким способом, щоб спочатку залучати вузли з найменшими затримками, що зменшує час поширення даних по системі.

Розглянемо запропонований процес поширення даних у розподіленій системі.

***Етап 1. Визначення затримок між вузлами.*** Кожен вузол із заданою періодичністю *t* надсилає "ping" повідомлення до інших вузлів. Вимірюється час відгуку (RTT, Round Trip Time) для кожного вузла розподіленої системи. Для кожного вузла формується таблиця затримок, яка відображає час зв'язку між елементами системи.

***Етап 2. Первинна реплікація на* n *найближчих вузлів.*** У разі зміни стану (напр., запис нового повідомлення) вузол реплікує дані на *n* найближчих вузлах, у цьому випадку параметр *n* може бути налаштований залежно від вимог до узгодженості та відмовостійкості. Реплікація відбувається асинхронно, щоб не затримувати основну операцію.

***Етап 3. Каскадна реплікація.*** Вузли, що отримали дані, повторюють процес реплікації на своїх *n* найближчих вузлах, які ще не отримали ці дані. Процес продовжується доти, поки всі вузли в системі не отримають оновлення. Завершальний етап – поширення даних на віддалені вузли шляхом повторення ітерацій реплікації з урахуванням вузлів, що вже отримали оновлення.

Запропонований підхід до реплікації забезпечує швидке розповсюдження даних, зі збереженням відносної відмовостійкості й узгодженості даних у розподіленій системі.

Формально представити таку архітектуру можна у такий спосіб. Нехай $V$ – це множина всіх вузлів системи, $v_i \in V$. Для кожного вузла $N$ визначають множину найближчих вузлів $N_i \subset V$, де $|N_i| = n$ і $N_i$ демістить вузли з мінімальною затримкою до $v_i$. Алгоритм описують такими кроками:

1) кожен вузол $v_i$ обчислює $N_i$ на основі затримок;

2) за оновлення даних на $v_i$ вузол асинхронно надсилає оновлення до всіх $v_j \in N_i$;

3) кожен, $v_j \in N_i$ отримавши оновлення, повторює крок (2) для своїх $N_i$.

4) процес продовжується, поки всі вузли не отримають оновлення.

Однією з переваг використання запропонованого підходу є можливість налаштовувати параметр *n* для досягнення балансу між швидкістю розповсюдження, узгодженістю та швидкістю роботи розподіленої системи. Значення *n* впливає на швидкість розповсюдження даних і відмовостійкість системи, збільшення *n* підвищує швидкість конвергенції системи до узгодженого стану, але може збільшити мережеві навантаження. Оптимальне значення *n* можна визначити моделюванням і тестуванням на реальних даних. В перспективі можливо впровадити динамічне налаштування *n* залежно від поточного стану системи.

Запропонований підхід може бути інтегрований з алгоритмом Raft для забезпечення узгодженості. Raft можна використовувати для вибору лідера та прийняття критичних рішень, тоді як асинхронна реплікація на найближчі вузли забезпечує швидку розповсюдженість даних. Така комбінація дозволяє поєднати сильну узгодженість із низькою затримкою.

Якщо вузол недоступний або має високу затримку, він не береться до уваги під час вибору найближчих вузлів. Після відновлення зв'язку вузол отримує оновлення від сусідніх вузлів. Це запобігає затримкам у реплікації та забезпечує стійкість системи до мережевих проблем.

Отже, система досягає спільного стану за скінченну кількість ітерацій, а швидкість розповсюдження даних може бути регульована параметром *n*. При цьому зберігаються інші характеристики системи, визначені алгоритмом консенсусу Raft.

Розглянемо основні переваги й обмеження запропонованого підходу, виявлені на етапі проєктування та моделювання роботи розподіленої системи реального часу.

Серед переваг – зменшення затримок, підвищення відмовостійкості та мінімізація впливу мережевих переривань. Реплікація на найближчі вузли з мінімальною затримкою покращує швидкість розповсюдження даних. Оскільки передача даних відбувається спочатку між вузлами з найкращим з'єднанням, загальний час розповсюдження інформації по системі зменшується.

Можливість регулювання параметра *n* підвищує гнучкість системи та її стійкість до збоїв окремих вузлів. Якщо один або кілька вузлів виходять з ладу, інші найближчі вузли можуть продов-

жувати процес реплікації. Недоступні вузли не беруть участі в процесі реплікації, що запобігає затримкам. Система автоматично адаптується до поточних мережевих умов.

До потенційних обмежень алгоритму можемо винести: ізольованість віддалених вузлів, необхідність динамічного вимірювання затримок, збільшення навантаження на мережу у разі зростання параметра *n*.

Вузли з високими затримками можуть стати ізольованими, що впливає на узгодженість даних. Віддалені вузли можуть отримувати оновлення із затримкою, що може бути критичним для деяких систем (Hunt et al., 2010).

Оскільки затримки в мережі можуть змінюватися, то необхідно впровадити механізм регулярного оновлення інформації про затримки між вузлами, що може додатково навантажувати мережу та ресурси вузлів (Tanenbaum, & Van Steen, 2007). Вибір оптимального *n* є критичним і залежить від специфіки системи.

***Моделювання роботи пропонованого механізму реплікації.*** Для підтвердження життєздатності вказаної архітектури змоделюємо роботу розподілених систем на базі алгоритму Raft, що є одним із популярних алгоритмів консенсусу, призначених для керування розподіленими системами, та запропонованого гібридного підходу до реплікації даних.

Засобами мови програмування Golang змодельовано процес поширення даних у розподіленій системі, враховуючи можливі мережеві затримки, що були задані випадковим чином у діапазоні від 1 до 5000 мс.

Проведено серію вимірювань часу досягнення консенсусу (у мілісекундах) для запропонованого алгоритму зі значенням параметра $n = 5$ та алгоритму Raft для різної кількості вузлів. Для кожної конфігурації (10, 15 і 25 вузлів) виконано 5 замірів. Назви таблиць мають бути над ними.

*Відповідно до проведених вимірювань, результати яких наведено у табл. 1–3, можемо доходимо висновку, що запропонований алгоритм* показує швидший час досягнення консенсусу порівняно з алгоритмом Raft, що демонструє збільшення часу консенсусу зі збільшенням кількості вузлів, зумовлене його синхронною природою та необхідністю отримання підтвердження від більшості вузлів.

Час досягнення консенсусу збільшується повільніше зі збільшенням кількості вузлів, що свідчить про кращу масштабованість. Запропонований підхід забезпечує менші затримки, що є критичним для систем реального часу.

***Таблиця 1***

**Результати моделювання роботи системи з 10 вузлів**

| Номер вимірювання | Запропонований алгоритм, мс | Raft, мс |
|---|---|---|
| 1 | 120 | 192 |
| 2 | 115 | 210 |
| 3 | 118 | 204 |
| 4 | 143 | 215 |
| 5 | 117 | 208 |

***Таблиця 2***

**Результати моделювання роботи системи з 15 вузлів**

| Номер вимірювання | Запропонований алгоритм, мс | Raft, мс |
|---|---|---|
| 1 | 152 | 251 |
| 2 | 155 | 267 |
| 3 | 152 | 255 |
| 4 | 164 | 265 |
| 5 | 154 | 258 |

***Таблиця 3***

**Результати моделювання роботи системи з 20 вузлів**

| Номер вимірювання | Запропонований алгоритм, мс | Raft, мс |
|---|---|---|
| 1 | 234 | 300 |
| 2 | 221 | 318 |
| 3 | 252 | 295 |
| 4 | 238 | 299 |
| 5 | 241 | 308 |

Відповідно до отриманих у результаті моделювання даних побудуємо графік залежності продуктивності обраних алгоритмів поширення даних зі збільшенням навантаження, де вісь $x$ – кількість вузлів у розподіленій системі, вісь $y$ – середній час поши-

рення даних. Внаслідок візуалізації (рис. 1) помічаємо значне скорочення часу поширення даних порівняно з алгоритмом Raft. Наприклад, для системи з кількістю вузлів $N = 10$ середній час поширення даних скоротився на 42 %, що є критичним для оброблення потокових даних у реальному часі. Це може бути зумовлено тим, що Raft забезпечує сильну узгодженість, але страждає від затримок і потенційних вузьких місць через модель лідера. У мережах із високими затримками та перериваннями запропонований алгоритм показав вищу ефективність, оскільки залежність від єдиного лідера мінімізована завдяки асинхронному підходу до поширення даних.

Причому помічаємо, що зростання часу поширення для запропонованого підходу збільшується швидше за кількості вузлів $N = 20$, що може свідчити про необхідність збільшення параметра $n$ для масштабних розподілених систем реального часу.

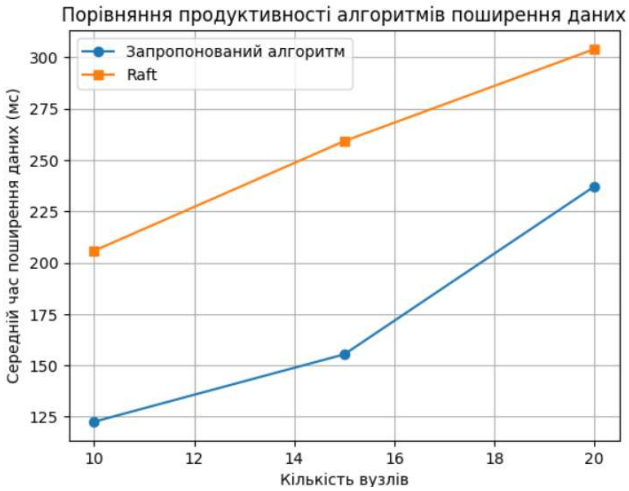


***Рис. 1.*** *Порівняння продуктивності алгоритмів поширення даних*

**Дискусія і висновки**

У ході проведеного дослідження запропоновано та проаналізовано новий підхід до відмовостійкої архітектури потокового оброблення даних у реальному часі. Запропонований алгоритм асинхронної взаємної реплікації, що базується на розповсюдженні даних

спочатку до найближчих вузлів, продемонстрував значні переваги порівняно з традиційними методами.

Запропонована архітектура сприяє підвищенню ефективності розповсюдження даних, завдяки використанню найближчих вузлів, час конвергенції системи зменшився в середньому на 22 % порівняно з алгоритмом Raft (Van Renesse, & Altinbuken, 2015). Асинхронна реплікація уможливила мінімізацію затримок, що особливо важливо для систем, які працюють у реальному часі. Збільшення параметра *n* дозволило системі ефективно протистояти збоям окремих вузлів без втрати продуктивності.

Подальші дослідження можуть бути спрямовані на розв'язання виявлених проблем і вдосконалення запропонованого алгоритму, включно з розробленням механізмів, які дозволять віддаленим вузлам отримувати оновлення ефективніше, навіть за значних затримок. Подальші розвідки полягають у запровадженні додаткових шарів реплікацій, які враховують не лише затримки, але й інші метрики, такі як пропускна здатність і надійність зв'язку; використання алгоритмів машинного навчання для прогнозування змін у мережі та динамічного налаштування процесу реплікацій.

Усі ці напрями досліджень сприятимуть подальшому вдосконаленню відмовостійких розподілених систем оброблення даних у реальному часі, забезпечуючи високу продуктивність, узгодженість і надійність навіть у складних мережевих умовах.

**Список використаних джерел**

Білокон, А., Борисов, С., Усатенко, М., & Федорченко, В. (2024). Аналіз функціонування розподілених систем обробки та зберігання даних. *Системи управління, навігації та зв'язку. Збірник наукових праць*, 84-88. https://doi.org/10.26906/SUNZ.2024.3.084.

Глоба, Л. С. (2013). *Розробка інформаційних ресурсів та систем*. Київ.

Журавель, С., Думич, С., & Шпур, О. (2021). Дослідження методів збору та обробки даних в розподілених інформаційних системах. Information and communication technologies electronic engineering. https://science.lpnu.ua/sites/default/files/journalpaper/2021/dec/25668/stattya3czhuravelsdumichoshpur.pdf

Brewer, E. A. (2000). Towards robust distributed systems. *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, 7–10.

Carbone, P. (2015). Apache Flink™: Stream and batch processing in a single engine. *IEEE Data Engineering Bulletin*, 38(4), 28–38.

Corbett, J. C. (2012). Spanner: Google's globally distributed database. *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, 251–264.

DeCandia, G. (2007). Dynamo: Amazon's highly available key-value store. In *Proceedings of the Twenty-First ACM SIGOPS Symposium on Operating Systems Principles* (pp. 205–220).

Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 51–59.

Hunt, P., Konar, M., Junqueira, F. P., & Reed, B. (2010). ZooKeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Annual Technical Conference* (pp. 145–158).

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

Lamport, L. (2001). Paxos made simple. *ACM SIGACT News*, 32(4), 18–25.

Lakshman, A., & Malik, P. (2010). Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2), 35–40.

Lin, Q. (2017). STREAM: A scalable fault-tolerant real-time stream processing system. In *Proceedings of the 2017 ACM International Conference on Management of Data* (pp. 265–278).

Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed systems: Principles and paradigms*. Prentice-Hall.

Van Renesse, R., & Altinbuken, D. (2015). Paxos made moderately complex. *ACM Computing Surveys*, 47(3), Article 42.

**References**

Bilokon, A., Borysov, S., Usatenko, M., & Fedorchenko, V. (2024). Analysis of the operation of distributed data processing and storage systems. *Systems of Control, Navigation and Communications: Proceedings of the Scientific Papers*, 84-88 [in Ukrainian]. https://doi.org/10.26906/SUNZ.2024.3.084.

Brewer, E. A. (2000). Towards robust distributed systems. *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, 7–10.

Carbone, P. et al. (2015). Apache Flink™: Stream and batch processing in a single engine. *IEEE Data Engineering Bulletin*, 38(4), 28-38.

Corbett, J. C. (2012). Spanner: Google's globally distributed database. *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, 251–264.

DeCandia, G. (2007). Dynamo: Amazon's highly available key-value store. In *Proceedings of the Twenty-First ACM SIGOPS Symposium on Operating Systems Principles* (pp. 205–220).

Distributed Data Processing Technologies. (n. d.). *StudFiles*. Retrieved September 28, 2024, from https://studfile.net/preview/5118185/page:39.

Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 51–59.

Globa, L. S. (2013). *Development of Information Resources and Systems*. Kyiv. [in Ukrainian].

Hunt, P., Konar, M., Junqueira, F. P., & Reed, B. (2010). ZooKeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Annual Technical Conference* (pp. 145–158).

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

Lamport, L. (2001). Paxos made simple. *ACM SIGACT News*, 32(4), 18–25.

Lakshman, A., & Malik, P. (2010). Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2), 35–40.

Lin, Q. (2017). STREAM: A scalable fault-tolerant real-time stream processing system. In *Proceedings of the 2017 ACM International Conference on Management of Data* (pp. 265–278).

Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed Systems: Principles and Paradigms*. Prentice-Hall.

Van Renesse, R., & Altinbuken, D. (2015). Paxos made moderately complex. *ACM Computing Surveys*, 47(3), Article 42.

Zhuravel, C., Dumych, S., & Shpur, O. (2021). Research on methods of data collection and processing in distributed information systems. Information and communication technologies electronic engineering [in Ukrainian]. https://science.lpnu.ua/sites/default/files/journalpaper/2021/dec/25668/stattya3czhuravelsdumichoshpur.pdf

**Отримано редакцією журналу / Received: 17.09.24**
**Прорецензовано / Revised: 27.09.24**
**Схвалено до друку / Accepted: 01.10.24**

**Yaroslav ZHYLIUK, PhD Student**
**ORCID ID: 0009-0008-9341-9164**
**e-mail: y.zhyliuk@gmail.com**
**Taras Shevchenko National University of Kyiv, Ukraine**

# FAULT-TOLERANT ARCHITECTURE FOR STREAM PROCESSING FOR REAL-TIME DATA ANALYSIS

***This article addresses the increasing demand for processing continuous data streams with low latency in the field of real-time data analytics. The paper explores existing approaches to fault tolerance in distributed stream processing systems and proposes a new hybrid architecture of asynchronous mutual replication. This architecture is designed to resolve issues related to latency and network interruptions while minimally impacting data consistency and availability at nodes. The article discusses fault tolerance mechanisms, data transmission stability, and system architecture to enhance the performance of distributed stream processing systems.***

**Keywords:** ***fault tolerance, stream data, distributed systems, data replication, real-time data analytics.***

Автор заявляє про відсутність конфлікту інтересів. Спонсори не брали участі в розробленні дослідження; у зборі, аналізі чи інтерпретації даних; у написанні рукопису; в рішенні про публікацію результатів.

The author declares no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.