COMPUTER SYSTEMS AND NETWORKS

Vol. 7, No. 1, 2025

DOI: https://doi.org/10.23939/csn2025.01.204

УДК 004.75

ОПТИМІЗАЦІЯ КОМУНІКАЦІЇ У ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ МЕТОДОМ FLAGBAG

Е. Є. Мальцев, О. В. Муляревич

Національний університет "Львівська політехніка",
кафедра електронних обчислювальних машин
E-mail: eduard.y.maltsev@lpnu.ua, oleksandr.y.muliarevych@lpnu.ua

© *Мальцев Е. Є, Муляревич О. В., 2025*

Наведено новий метод оптимізації серіалізації даних для міжсервісної комунікації у розподілених системах, названий FlagBag. Запропонований метод спрямований на зменшення затримки передачі даних між сервісами шляхом впровадження ефективної організації структури даних та алгоритму їх серіалізації. Дослідження проведено з використанням Apache Avro як базового формату для порівняння. Результати експериментів показали, що FlagBag зменшує середню затримку передачі даних між сервісами на 18 %, порівняно з немодифікованим Avro, а обсяг переданих даних скорочується на 15 % в окремих випадках. Крім того, запропонований метод демонструє стабільну продуктивність під час збільшення розміру повідомлень до 10 кБ, забезпечуючи перевагу у середньому на 15 % у часі передачі за такого сценарію. Розглянуто аспекти інтеграції FlagBag в наявні архітектури мікросервісів, включаючи потенціал зниження операційних витрат на підтримку сервісів у високонавантажених системах. Проведені тести продуктивності підтвердили переваги методу в умовах реального навантаження, що робить FlagBag перспективним рішенням для розв'язання задач із високими вимогами до швидкості та ефективності міжсервісної комунікації. Запропонований підхід є універсальним і може бути адаптований для інших форматів серіалізації, забезпечуючи покращення продуктивності в широкому спектрі застосувань.

Ключові слова: передача даних, кодування, обмін інформацією, протоколи, оцінка ефективності.

Вступ

З розвитком розподілених обчислювальних систем, обмін даними між сервісами набув ключового значення для забезпечення їх ефективної роботи. Однією з головних задач є оптимізація процесів серіалізації та десеріалізації даних, які значно впливають на загальну продуктивність системи. У сучасних мікросервісних архітектурах ці процеси стають особливо критичними через велику кількість запитів між сервісами у реальному часі.

Відомі методи серіалізації, такі як Protocol Buffers, Apache Avro чи MessagePack, демонструють високий рівень ефективності, проте вони не завжди здатні забезпечити оптимальну продуктивність у специфічних умовах, таких як висока частота обміну невеликими обсягами даних або необхідність роботи у чутливих до затримок системах. Це спонукало нас до розробки нового методу, який отримав назву FlagBag.

Метод FlagBag пропонує особливий підхід до серіалізації даних, орієнтований на мінімізацію затримок у міжсервісній комунікації. Його унікальність полягає у використанні спеціальних структур

для зменшення обсягу даних, що передаються, без втрати інформаційного змісту. У статті ми детально розглядаємо концепцію методу FlagBag, його архітектурні особливості, переваги порівняно з відомими підходами, а також результати експериментальних досліджень, які підтверджують його ефективність, порівнюючи з альтернативними підходами.

У цьому дослідженні ми хочемо продемонструвати потенціал методу FlagBag для покращення продуктивності комунікації та зменшення затримок у розподілених системах, а також зробити внесок у розвиток сучасних технологій серіалізації.

Огляд літературних джерел

Щоб краще зрозуміти поточний стан речей, пропонуємо розглянути сучасні наукові роботи та інші літературні джерела, пов'язанні з нашою темою.

Дослідження [1] демонструє, що метод пакування бітів підвищує продуктивність у 2,5 раза порівняно з традиційними горизонтальними методами, значно оптимізуючи операції обробки даних. Це зменшує вимоги до зберігання приблизно на 30 %, забезпечуючи більш ефективне використання простору для наборів даних. Крім того, цей метод зменшує час виконання запиту приблизно на 40 %, підкреслюючи його вплив на підвищення швидкості та ефективності пошуку даних.

Дослідження [2] використовує методи пакування бітів для сейсмічних даних, що є цікавим у контексті оптимізації для нашого дослідження. Дослідження демонструє, що бітове пакування є компактним кодуванням, яке знижує витрати на зберігання та передачу. Його результати, зокрема зменшення використання місця у сховищах даних на 20 % і ефективна швидкість стиснення, підкреслюють важливість мінімізації накладних витрат, подібно до цілей FlagBag в розподілених системах.

У статті [3] показано, що метод упаковки бітів у ґратчастий квантовій хромодинаміці досягають до 100 разів більшої точності, порівняно зі стандартними 16-бітними форматами, без значного збільшення розміру сховища.

У ньому підкреслюється, що 20-бітні та 30-бітні формати забезпечують оптимальний компроміс між точністю та продуктивністю, досягаючи значної ефективності зберігання.

У статті [4] демонструється, як кодування, як-от DFE та EDFE, покращують продуктивність, при цьому операції сканування досягають прискорення в 1,47 раза, а операції вибірки - у 1,33 раза. Зменшуючи кількість оброблених бітів і дозволяючи ранню зупинку, ці кодування зменшують накладні витрати, підтримуючи сумісність із сучасними форматами зберігання.

Стаття [5] про CodecDB демонструє, як стовпцеві бази даних з підтримкою кодування перевершують традиційні системи, які не за архітектурою не помічають кодування даних. CodecDB досягає на 40 % кращого коефіцієнта стиснення та вибирає оптимальне кодування з точністю 90 %, підвищуючи ефективність зберігання. Продуктивність запитів підвищується шляхом операторів, специфічних для конкретного кодування, досягаючи 10-кратного прискорення за тестом TPC-H і 3-кратного прискорення за тестом Star-Schema.

У статті [6] наведена бітова схема для зіставлення шаблонів регулярних виразів, яка стискає представлення позиційних автоматів внаслідок використання їх однорідних властивостей і практичних особливостей регулярних виразів. Цей підхід значно зменшує використання пам'яті, порівняно з двома поширеними представленнями скінченних автоматів, і перевершує сучасний движок RE2 для великих скомпільованих регулярних виразів.

Стаття [7] вводить кодування спадного бітового пакування для даних частотної області для розв'язання проблем неефективності в точності та перекошеному розподілі значень. Завдяки кількісній оцінці даних на основі співвідношення сигнал/шум (SNR) і динамічному зменшенню бітової ширини за допомогою сортування метод досягає значної ефективності зберігання. Розгорнутий в Apache IoTDB, він підтримує як пряме частотне кодування домену, так і стиснення домену з втратами.

Стаття [8] зосереджена на вдосконаленні управління даними IoT та енергоефективності за допомогою гібридного методу стиснення, що поєднує Delta Encoding та Run-Length Encoding (RLE). Цей підхід забезпечує неймовірний ступінь стиснення 99,25 %, а час стиснення становить лише 139 мікросекунд, що значно зменшує розмір даних та витрати на зберігання. Мінімізуючи вимоги до передачі та зберігання даних, цей метод також сприяє економії енергії, що робить його добре придатним для додатків IoT з обмеженими ресурсами.

Стаття [9] представляє Bit-Block Compressed Sparse Row (B2SR), дворівневий формат бітового пакування для матриць суміжності в графічній аналітиці, що забезпечує зменшення використання місця в 32 рази та значне прискорення на графічних процесорах.

Оцінки графічних процесорів NVIDIA Pascal і Volta показують покращення до 40× і 6555× для SpMV і SpGEMM відповідно, і до 433× прискорення для BFS і 69× для підключених компонентів. Використовуючи специфічні для GPU можливості маніпулювання б ітами т а глибоке навчання з підкріпленням (DRL) для оптимізації розмірів блоків, B2SR демонструє ефективну обробку розріджених двійкових матриць.

У цьому [10] документі пропонується загальний підхід pushdown предикатів для покращення продуктивності запитів у стовпцевих системах зберігання, таких як Apache Parquet, шляхом зниження витрат на декодування. Використовуючи інструкції з маніпулювання бітами в архітектурі X86, цей підхід дає змогу безпосередньо вибирати закодовані значення без повного декодування, що значно підвищує ефективність. Експерименти показують покращення продуктивності запитів до 10× для запитів сканування паркету та до 5,5× - для складних наскрізних запитів у Apache Spark. Ці висновки узгоджуються з нашим дослідженням FlagBag, ілюструючи, як оптимізація на рівні бітів може прискорити доступ до даних та їх обробку в сучасних аналітичних системах.

У статті [11] представлена система потокової передачі подій з високою роздільною здатністю з використанням двійкового детектора з розрідженим зчитуванням і формату Centroided Event Stream (CES). Завдяки стисненій розрідженій структурі блока система досягає повнокадрової роздільної здатності 382 мкс і масштабується до 24 мкс для невеликих областей, значно зменшуючи накладні витрати на зберігання.

У статті [12] досліджується ефективне розгортання двійкових нейронних мереж (BNN) на мікроконтролерах з обмеженими ресурсами, оптимізуючи використання пам’яті шляхом адаптації параметрів шару до специфікацій мови C. Незалежна від платформи бібліотека C для шарів BNN була розроблена та протестована на мікроконтролері Cortex-M7 для роботизованої класифікації дотиків, досягнувши покращення точності на 2 %, підвищення ефективності пам’яті - на 15 % та зниження затримки - на 25 %.

Методологію цього дослідження, архітектуру, взаємодію компонентів та технологію вимірювання результатів було імплементовано на основні нашого дослідження [13]. Apache Avro як базовний формат для дослідження та фактори для його вибору описані у дослідженні [14], в ньому ми також розглянули вплив цього формату на розмір серіалізованого представлення.

Постановка задачі

Метою дослідження є визначення можливості покращення затримки міжсервісної комунікації щанайменше на 10 %, порівняно з Apache Avro, шляхом впровадження методу оптимізації серіалізації атрибутів булевого типу FlagBag.

Нульова гіпотеза, наведена у табл. 1, означає що метод FlagBag не забезпечує мінімального зниження затримки передачі даних (μ) між сервісами на 10 % порівняно з немодифікованою серіалізацію AVRO (RAE). Інакше кажучи, різниця між середніми значеннями затримки є меншою або дорівнює 10 %. Альтернативна гіпотеза означає що метод FlagBag забезпечує зниження затримки передачі даних між сервісами на 10 % або більше порівняно з AVRO. Для статистичного аналізу також буде впроваджена змінна NRAE = $0.9\mu_{\mathrm{RAE}}$, це довзволить легко перевірити, чи є різниця на 10 % статистично значущою.

Таблиця 1

Сформовані гіпотези на основні попереднього дослідження

Нульова гіпотеза (H_0)	Альтернативна гіпотеза (H_1)
Змінна $0.9\mu_{\text{RAE}}$ (секунди) має менші або рівні значення, ніж змінна μ_{FlagBag} $H_0: \mu_{\text{FlagBag}} \geq 0.9\mu_{RAE}$	Змінна $0.9\mu_{\text{RAE}}$ має більші значення, ніж змінна μ_{FlagBag}. $H_1: \mu_{\text{FlagBag}} < 0.9\mu_{RAE}$

Щоб провести достовірні вимірювання в тестовому середовищі з можливістю подальшого відтворення, потрібно виконати такі завдання:

· встановити тестове, ізольоване середовище, яке описане в [13] на платформі Google Cloud;

· адаптувати категорії, які перевіряються для 15 наборів даних, з кожним з форматів серіалізації (FlagBag, RAE);

· запустити тест на виконання та збір метрики затримки (RAE Latency seconds та FlagBag Latency seconds) за сценарієм міжсервісної взаємодії, описаним в [13];

· провалідувати гіпотезу за допомогою одностороннього тесту за t-критерієм Стьюдента для парних вибірок;

· проаналізувати та описати результати тесту.

Серіалізація та її важливість у міжсервісній комунікації

Серіалізація - це процес перетворення об’єкта або структури даних у формат, який можна легко зберігати або передавати. Основна мета серіалізації полягає у тому, щоб зробити дані придатними для передачі через мережу або збереження у файлі, щоб їх пізніше можна було відновити (десеріалізувати) до початкової форми. Цей процес є критичним для міжсервісної комунікації в розподілених системах, де дані повинні передаватися між різними компонентами, часто реалізованими на різних платформах або мовах програмування.

У контексті мікросервісної архітектури сервіси часто взаємодіють через мережу, використовуючи різні формати даних. Серіалізація дає змогу ефективно кодувати складні структури даних у потік байтів, який можна передати іншим сервісам. Після отримання цього потоку сервіс-дестинатор виконує десеріалізацію для відновлення початкових даних. Вибір потрібного формату серіалізації може значно вплинути на продуктивність системи, оскільки різні формати мають різні характеристики щодо швидкості та розміру даних.

Існує багато популярних форматів серіалізації, серед яких варто виокремити JSON, Protocol Buffers, Apache Avro, MessagePack та інші. JSON є одним з найпоширеніших форматів, оскільки його легко читає людина і широко підтримується в різних мовах програмування. Проте його недоліком є порівняно великий розмір даних та менша ефективність порівняно з бінарними форматами. Бінарні формати, такі як Protocol Buffers або Avro, забезпечують більшу компактність та швидкість, що робить їх більш привабливими для систем з високими вимогами до продуктивності.

Процес серіалізації можна поділити на кілька етапів. Спочатку об’єкт даних перетворюється у структурований формат, який може бути збережений або переданий. Далі цей структурований формат кодується у потік байтів, що дозволяє легко передати його мережею. На стороні отримувача відбувається зворотний процес - десеріалізація, яка передбачає декодування потоку байтів та відновлення початкового об’єкта. Ці операції можуть бути витратними з погляду часу і ресурсів, особливо якщо розмір даних великий або якщо система працює в реальному часі.

Серіалізація також відіграє ключову роль у масштабованості системи. Вибір відповідного формату може значно знизити навантаження на мережу і сервери, оскільки ефективно серіалізовані дані займають менше місця і швидше передаються. Наприклад, використання Protocol Buffers дає

змогу скоротити розмір переданих даних у середньому на 30- 40 % порівняно з JSON, що важливо для великих систем, де навіть незначні оптимізації можуть мати великий вплив на загальну продуктивність.

Наприклад, у типовій ситуації, відображеній на рис. 1, серіалізація використовується для передачі об'єкта типу "повідомлення" від сервісу А до сервісу В. Сервіс А серіалізує об'єкт у бінарний потік, який надсилається мережею. Сервіс В отримує цей потік, виконує десеріалізацію та отримує початковий об'єкт для подальшої обробки. У такому контексті ефективність серіалізації прямо впливає на загальний час обробки повідомлень і, як наслідок, на швидкість відповіді сервісу.

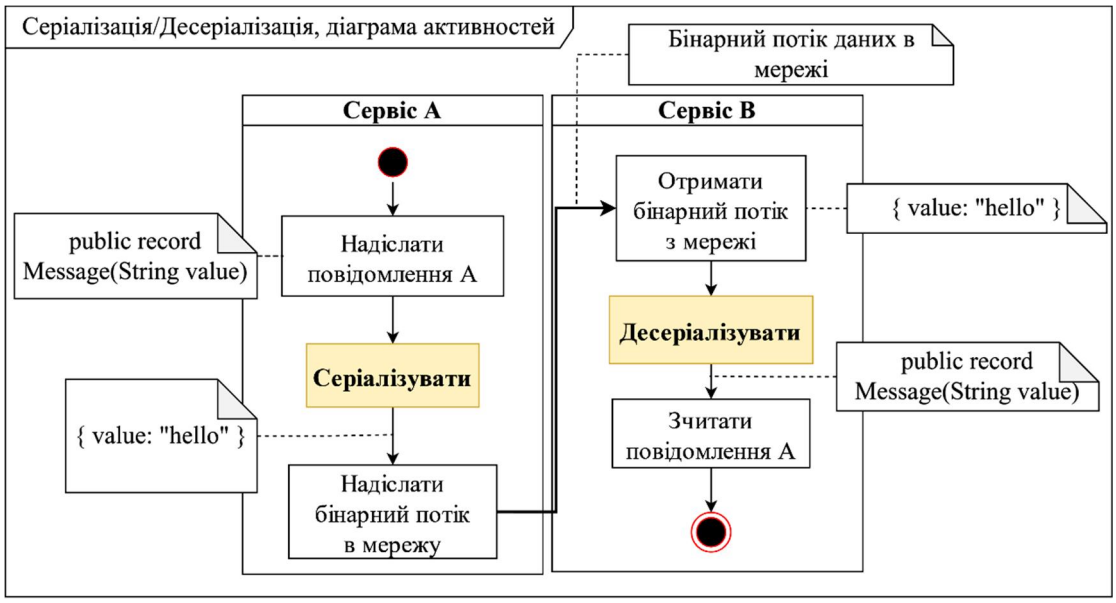

Рис. 1. Серіалізація у типовій послідовності взаємодії сервісів

Використання пакування бітів у суміжних галузях науки

Схожі методи вже широко застосовуються для ефективного збереження та представлення в пам'яті набору булевих значень або двійкових індикаторів стану. Таке представлення вважається компактним та швидким, особливо в системах чутливих до затримок, таких як ігрові двигуни типу Unreal Engine. Фактична імплементація зазвичай схована за абстракцією мови програмування, що робить їх зручними для використання.

На рис. 2 видно, як створюють структуру з наперед заданими індикаторами стану, таких як TALL, DARK та інші. Для збереження стану використовуються біти у виділеному просторі **int32,** які займають 32 біти (4 байти). Це значить, що ми можемо представити 32 булевих індикатори стану (0 або 1) за допомогою одного поля типу **int32**. Внутрішній механізм цієї структури дає змогу записати та зчитати значення конкретного біту. Приклад використання такої структури у методі класу наведено на рис. 3.

```
1    UENUM(BlueprintType, meta = (Bitflags, UseEnumValuesAsMaskValuesInEditor = "true"))
2    enum class ECharacterQualities: uint8
3    {
4       NONE       = 0 UMETA(Hidden),
5       TALL       = 1 << 0,
6       DARK       = 1 << 1,
7       HANDSOME   = 1 << 2,
8       PROGRAMMER = 1 << 3,
9    };
10   ENUM_CLASS_FLAGS(ECharacterQualities);
```

Рис. 2. Оголошення оптимізованої структури даних в C++

```
1 UFUNCTION(BlueprintCallable)
2 bool IsTall() const
3 {
4   return QualityFlags & ECharacterQualities::TALL;
5 }
```

Рис. 3. Приклад використання оптимізованої структури даних в C++

Використання у системах управління базами даних

Схожі підходи використовуються й у системах баз даних, таких як PostgreSQL і MySQL. Кодування з бітовим прапорцями - це техніка, яка використовується для ефективного зберігання кількох логічних атрибутів в одному цілочисельному стовпці. Цей метод особливо корисний для представлення індикаторів стану. Наприклад, можна визначити стан прапорця в такому типі за допомогою запиту:

SELECT (flags & 1) = 1 AS is_enabled FROM your_table; (1)

Операцію встановлення значення індикатора можна здійснити, встановлюючи 2-й біт в значення 1 за допомогою запиту:

UPDATE your_table SET flags = flags | (1 << 2) WHERE condition; (2)

Операцію вимкнення індикатора можна здійснити так:

UPDATE your_table SET flags = flags & ~ (1 << 2) WHERE condition; (3)

Для зручності використання зазвичай використовують ще допоміжні функції, такі як set_flag, як показано на рис. 4.

```
1    CREATE OR REPLACE FUNCTION set_flag(flags INT, position INT) RETURNS INT AS $$
2    BEGIN
3    RETURN flags | (1 << position); -- Set the specified bit
4    END;
5    $$ LANGUAGE plpgsql;
```

Рис. 4. Допоміжна SQL функція для керування бітовими індикаторами

Ця функція дає змогу спростити процес встановлення індикатора за індексом 2, наприклад так:

UPDATE your_table SET flags = set_flag(flags, 2) WHERE id = 1; (4)

У базах даних такий підхід використовуються для збереження місця на диску, оперативній пам’яті та для покращення продуктивності мережної комунікації.

Використання у вбудованих системах та IoT

Вбудовані системи, такі як ті, наприклад, що використовують мікроконтролери Arduino або ARM, часто використовують кодування бітовими прапорцями для ефективного управління булевими параметрами. Цей підхід зумовлений обмеженою пам’яттю та обчислювальною потужністю, доступними в цих системах. Один байт або слово ділиться на біти, де кожен біт представляє булевий стан. Далі наведений типовий приклад використання бітової стрічки розміром 8 біт (байт).

- Біт 0: датчик активний (is_sensor_active).
- Біт 1: Виявлено помилку (is_error_detected).
- Біт 2: Пристрій приєднано до живлення (is_powered).
- Біт 3: Зв’язок увімкнено (is_communication_enabled).
- Біти 4-7: Зарезервовано для майбутнього використання або інших прапорців.

Керування статусами в Arduino може виконуватись так:

$$flags \mid= (1 << IS\_SENSOR\_ACTIVE); \quad (5)$$

$$flags \mid= (1 << IS\_POWERED); \quad (6)$$

Тестування стану для конкретного біту може виконуватись так:

$$if\ (flags\ \&\ (1 << BIT\_POSITION))\ \{\ //\ Біт\ ввімкнений\ \} \quad (7)$$

Використання у мережних протоколах

Прапорці можуть бути упаковані в один байт або слово для передачі через протоколи зв'язку, такі як I²C, PPI або UART, зменшуючи розмір та покращуючи ефективність використання мережі. У протоколі TCP/IP бітові прапорці мають вирішальне значення для обміну даними між пристроями через мережу. Ці прапорці є компактними представленнями інформації, яка використовується для керування та моніторингу стану TCP-з'єднання. Прапорці є частиною заголовка TCP, який передається з кожним пакетом. Це дає змогу зменшити розмір серіалізованого представлення та затримки в мережі. Заголовок TCP містить 16-бітне поле для прапорців (також відомих як контрольні біти), які слугують булевими індикаторами конкретних станів або дій, що вимагаються протоколом, наприклад:

16-bit indicators: URG ACK PSH RST SYN FIN *Rest (8)

Кожен біт відповідає певному логічному стану:

- URG: Поле термінового покажчика є важливим.
- ACK: Поле визнання є важливим.
- PSH: Функція push (дані повинні бути негайно відправлені в додаток).
- RST: скиньте з'єднання.
- SYN: синхронізація порядкових номерів (використовується для ініціювання з'єднання).
- FIN: Більше немає даних від відправника (замикання з'єднання).
- *Rest: Інші біти в 16-бітному полі зарезервовані для майбутнього використання або розширень (наприклад, інші опції з'єднання).

Використання в стовпцевих форматах серіалізації

Parquet - це популярний стовпцевий формат серіалізації, який використовує декілька методів кодування, зокрема бітове пакування, для мінімізації зберігання та оптимізації продуктивності запитів, але, бо цей формат є стовпцевим, він рідко використовується для обміну даним в міжсервісній комунікації.

Використання в операційних системах

В Linux і Unix-подібних операційних системах дозволи на файли є хорошим прикладом того, як кодування бітових прапорів використовується для компактного керування кількома логічними станами. Дозволи в Linux контролюють доступ до файлів і каталогів та кодуються за допомогою фіксованого набору бітових прапорців у 9- або 12-бітному форматі, приклад представлення дозволів може бути таким: "rw-r--r--".

Таблиця 2

Використання пакування бітів в UNIX ОС для збереження метаданих

Власник	Приклад дозволів	Бінарне представлення	Десяткове представлення
Owner	rw- (read and write).	110	6
Group	r-- (read only).	100	4
Others	r-- (read only).	100	4

Три біти для кожної групи упаковуються в одну вісімкову цифру, створюючи компактне 9-бітне представлення. Таку логіку зберігання метаданих можна легко пояснити тим, що ми хочемо мінімізувати витрати місця на кожен файл, і як ми бачимо, це працює добре.

Використання в інших форматах серіалізації

Cap'n Proto зберігає булеві поля за допомогою ефективного механізму упаковки бітів, де кожне логічне значення наведено у вигляді одного біта в межах 64-бітного слова. Цей підхід гарантує, що кілька булевих полів щільно упаковані разом у пам'яті, мінімізуючи накладні витрати на зберігання. Індикатори стану призначаються конкретним бітовим позиціям на основі ідентифікаторів полів у схемі, при цьому нижчі ID зіставляються з нижчими бітами. Під час читання або запису булевого значення Cap'n Proto використовує побітові операції для вилучення або зміни відповідного біта, не впливаючи на інші в тому самому слові. Якщо структура містить понад 64 логічних значень, виділяються додаткові 64-бітові слова, зберігаючи компактність і визначене розміщення. Така конструкція дає змогу Cap'n Proto обробляти логічні поля з високою ефективністю як з погляду використання пам'яті, так і швидкості передачі. На рис. 5 наведено приклад схеми Cap'n Proto, яка включає логічні поля, які за замовчуванням зберігаються шляхом бітового пакування.

```
1    struct DeviceStatus {
2        isOnline @0 :Bool;
3        hasError @1 :Bool;
4        isCharging @2 :Bool;
5        needsUpdate @3 :Bool;
6        isLowBattery @4 :Bool;
7        isLocked @5 :Bool;
8    }
```

Рис. 5. Приклад використання бітових прапорців у форматі Cap'n Proto

Для читання логічних полів у Cap'n Proto за допомогою коду зазвичай використовується API генерації коду, наданий компілятором схеми Cap'n Proto (компілятор capnp). Приклад команди генерації коду в Cap'n Proto:

capnp compile -oc++ DeviceStatus.capnp (9)

```
1    // Доступ до кореневої структури
2    DeviceStatus::Reader deviceStatus = message.getRoot<DeviceStatus>();
3
4    // Простий доступ до булевих статусів
5    bool isOnline = deviceStatus.getIsOnline();
6    bool hasError = deviceStatus.getHasError();
7    bool isCharging = deviceStatus.getIsCharging();
8    bool needsUpdate = deviceStatus.getNeedsUpdate();
9    bool isLowBattery = deviceStatus.getIsLowBattery();
10   bool isLocked = deviceStatus.getIsLocked();
```

Рис. 6. Представлення об'єкта, згенерованого зі схеми Cap'n Proto, та його використання

На рис. 6 наведений згенерований об'єкт та його використання. Це показує, що використання бітових стрічок для покращення різних характеристик розподілених систем використовують широко і давно в суміжних галузях комп'ютерних наук, але саме в архітектурах, основаних на обміні повідомлень, він представлений досить обмежено, відповідно, виникає питання, чи можливо використати цей підхід для кодування повідомлень в міжсервісній комунікації й так покращити такі характеристики, як затримка передачі даних, пропускну здатність мережі, використання місця на диску тощо.

Логічні типи AVRO та їх застосування в імплементації методу FlagBag

У міжсервісному зв'язку на основі Java логічні типи Apache Avro покращують серіалізацію, поєднуючи компактність двійкових даних із семантичним багатством, необхідним для бізнес-логіки. Логічні типи визначаються в схемі Avro за допомогою анотації logicalType на примітивних типах, таких як int, long або bytes, без зміни двійкового представлення даних. Наприклад, логічний тип date використовує int для зберігання кількості днів з моменту епохи Unix (1 січня 1970 року), а під час десеріалізації в Java бібліотека Avro відображає це в java.time.LocalDate. Аналогічно логічний тип timestamp-millis використовує long для зберігання мілісекунди з моменту епохи, які бібліотека перетворює в java.time.LocalDateTime або java.time.Instant на основі вимог програми. Для десяткового дробу Avro кодує немасштабоване значення в байтах або фіксованому форматі та використовує визначену схемою точність і масштаб для десеріалізації його в java.math.BigDecimal.

Процес серіалізації в Java передбачає генерацію класів зі схеми Avro за допомогою таких інструментів, як плагін Avro Maven. Ці класи реалізують інтерфейс SpecificRecord і включають вбудовані методи для серіалізації та десеріалізації, такі як методи serialize() та deserialize(), які автоматично обробляють логічні типи. Логіка десеріалізації використовує метадані схеми для інтерпретації необроблених двійкових даних у відповідні типи Java. Наприклад, під час десеріалізації бібліотека Avro використовує утиліту LogicalTypes, щоб зіставити визначений схемою логічний тип з відповідним перетворювачем. Об'єкт LogicalTypes.date() відображає int з LocalDate, тоді як Logical Types.timestampMillis() відображає LocalDateTime. Ці перетворення гарантують, що розробники можуть працювати з високорівневими абстракціями та об'єктами у своїх програмах, а базова комунікація залишається ефективною. Логічні типи також дають змогу точно застосовувати схеми та відповідну валідацію під час міжсервісного зв'язку, що має вирішальне значення для забезпечення узгодженості даних і запобігання невідповідності типів у розподілених системах. Валідація схеми Avro гарантує, що серіалізовані дані відповідають схемі, визначеній відправником, а процес десеріалізації гарантує, що сервіси можуть правильно інтерпретувати дані.

Можливості еволюції схеми, такі як додавання нових полів зі значеннями за замовчуванням або зміна необов'язкових полів, бездоганно працюють з логічними типами, за умови, що базовий примітивний тип залишається незмінним. Наприклад, логічний тип timestamp-millis, закодований як long, залишається сумісним у різних версіях, дозволяючи службам, що використовують старіші версії схеми, обробляти дані без помилок.

З погляду продуктивності логічні типи Avro оптимізують як розмір серіалізації, так і швидкість обробки. Оскільки дані зберігаються у необробленій двійковій формі (наприклад, int для дати або long для часової позначки), розмір корисного навантаження мінімальний, що зменшує вимоги до пропускної здатності для міжсервісного зв'язку.

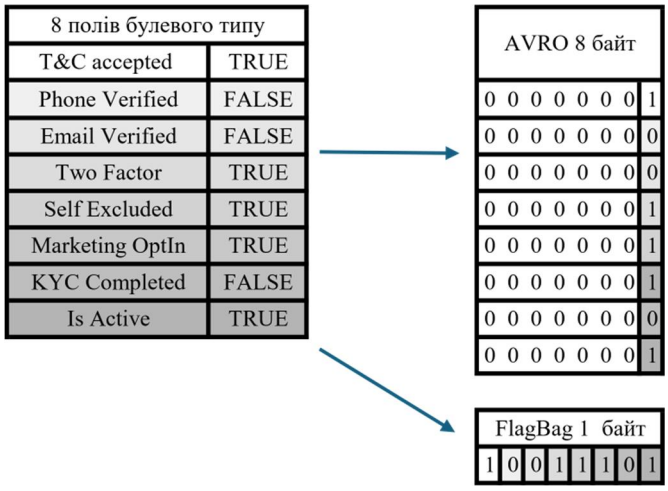

Рис. 7. Порівняння серіалізованого представлення булевих властивостей для RAE та FlagBag

Під час виконання вартість перетворення між двійковими даними та рідними типами Java є незначною порівняно з перевагами компактного сховища та ефективністю мережі. Зважаючи на описані переваги та модульність, інтеграція FlagBag в AVRO була імплементована саме за допомогою логічних методів, що дозволяє зберігати прапорці більш компактно, що зменшує використання дискового простору, а також прискорює передачу серіалізованих даних. На рис. 7 можна побачити ілюстрацію різниці використаного простору методом FlagBag та RAE.

Результати дослідження

Під час дослідження було розроблено та експериментально перевірено метод серіалізації даних булевого типу FlagBag, орієнтований на зниження затримок міжсервісної комунікації в розподілених системах. Основні результати дослідження підтверджують оригінальність та ефективність запропонованого підходу. FlagBag використовує оптимізовану структуру даних із маркерами стану (прапорцями), що дає змогу скорочувати обсяг переданих даних без втрати точності. FlagBag зосереджується на оптимізації для сценаріїв високочастотного обміну малими обсягами даних, що є поширеною проблемою у мікросервісних архітектурах, чутливих до затримок. Експериментальні дослідження проводились на тестовій розподіленій системі [13], де було порівняно FlagBag та Apache Avro за показником загальної затримки виконання розподіленого сценарію.

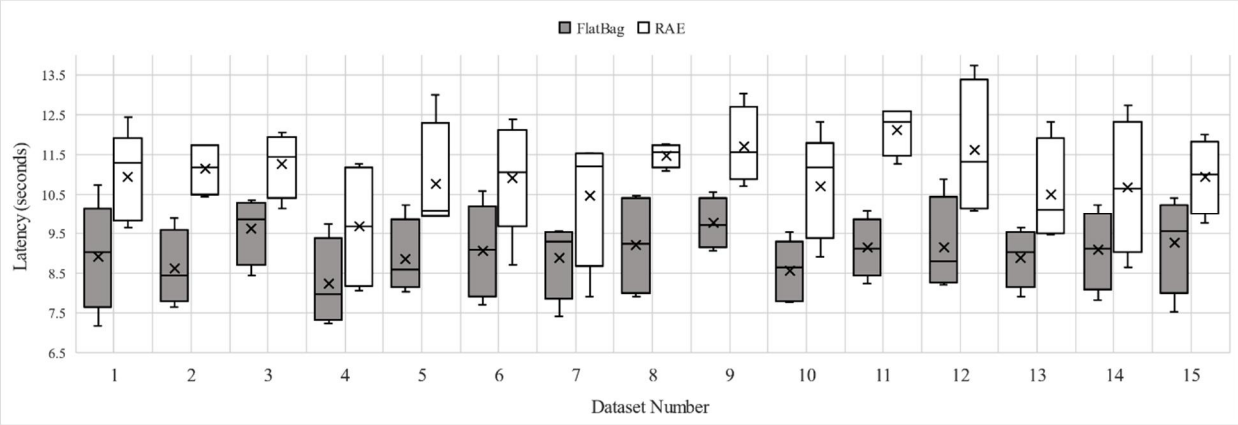

Рис. 8. Візуалізація експериментальних результатів розподіленої затримки RAE та FlagBag

Таблиця 3

Загальна затримка в секундах та загальний розмір набору даних для FlagBag та RAE

Формат	Середня затримка	Загальний розмір (кбайт)
FlatBag	9.01	1110000
RAE	10.99	1300000
Зменшення (%)	**18**	**15**

Таблиця 4

Результати експериментального порівняння методів FlagBag та RAE (сценарій 1-15)

№	Середня затримка FlagBag (секунди)	Середня затримка RAE (секунди)	Зменшення затримки (%)	Серед. розмір повідомлення (байт)
1	2	3	4	5
1	8.92	10.95	18.54	500
2	8.61	11.14	22.71	1000

Продовження табл. 4

1	2	3	4	5
3	9.63	11.27	14.55	2000
4	8.23	9.68	14.98	3000
5	8.86	10.77	17.73	4000
6	9.07	10.93	17.02	5500
7	8.9	10.46	14.91	6000
8	9.22	11.48	19.69	6500
9	9.76	11.71	16.65	7000
10	8.57	10.72	20.06	7500
11	9.14	12.12	24.59	8000
12	9.16	11.62	21.17	8500
13	8.9	10.5	15.24	9000
14	9.08	10.67	14.90	9500
15	9.27	10.93	15.19	10000

Як показано у табл. 3, середня затримка для FlagBag виявилася на 14- 22 % меншою, ніж для Apache Avro, залежно від сценаріїв навантаження. Згідно з табл. 4, метод FlagBag зменшив середній обсяг переданих даних на 15 %, що позитивно впливає на мережну продуктивність.

Метод FlagBag був легко інтегрований у тестову мікросервісну платформу за допомогою логічних типів, що підтверджує його практичну придатність.

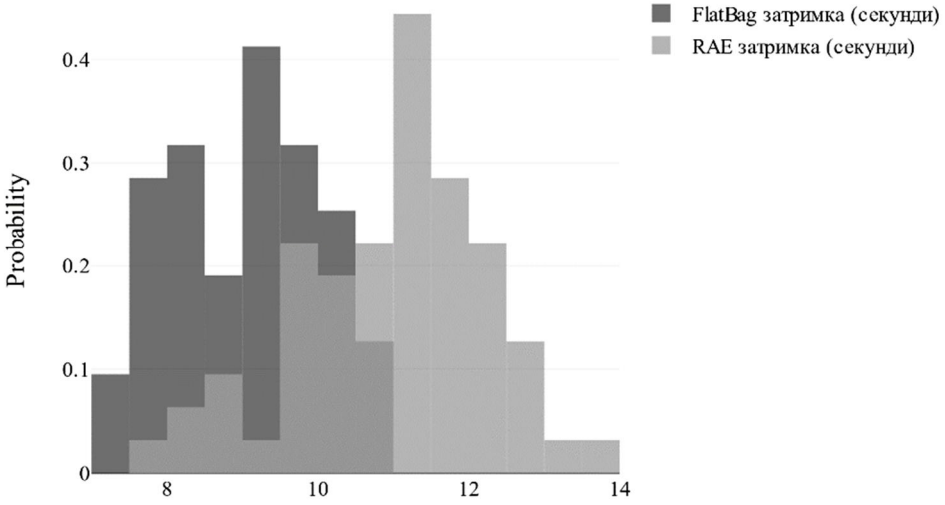

Рис. 9. Гістограма ймовірностей для розподіленої затримки FlagBag та RAE

На рис. 9 зображено розподіл імовірностей затримок для FlatBag та RAE. Затримки для FlatBag та RAE нормально розподілені, але з деякими відмінностями в розподілі, затримки RAE мають ширший розподіл, порівняно з FlagBag, що може свідчити про більшу варіабельність. FlatBag має яскравий пік приблизно 9 секунд, для RAE основний пік - приблизно 11 секунд.

Загалом затримки FlagBag є нижчими, ніж затримки RAE, особливо в областях після 11 секунд, де частота затримок RAE помітно збільшується, а FlagBag, навпаки, зменшується.

На рис. 10 відображено залежність між затримкою у секундах для FlatBag та RAE. З графіка можна зробити кілька висновків, по-перше, збільшення часу відповіді для FlatBag затримки, здається,

корелює зі збільшенням часу відповіді для затримки RAE. Це означає, що коли затримка в одному вимірі зростає, то вона також зростає і в іншому, що може бути пов'язано з однаковими умовами тестування для кожного сценарію. Дані демонструють деяке розсіювання, особливо у разі затримок FlatBag більше ніж 9 секунд.

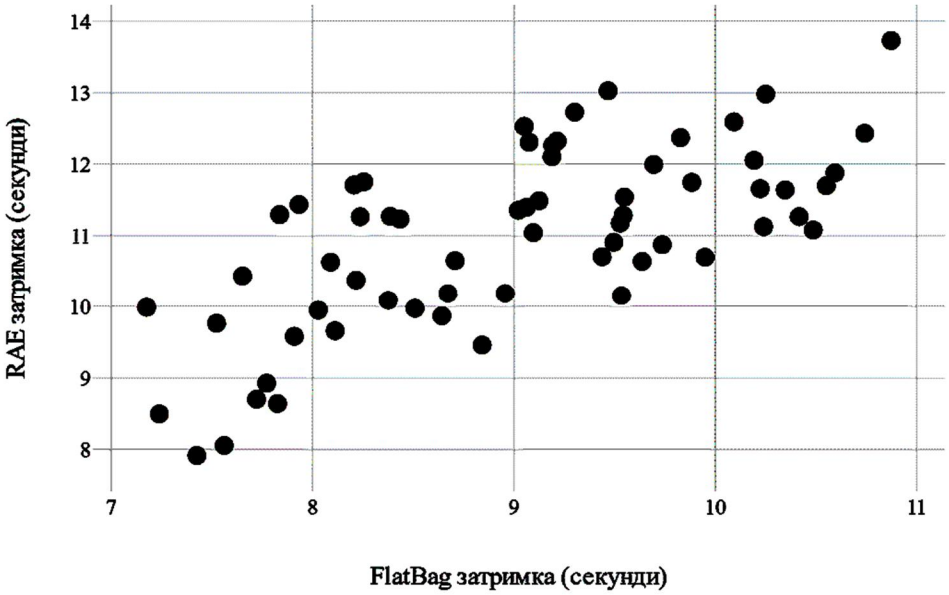

Рис. 10. Діаграма розсіювання порівняння серіалізованого представлення для булевих полів для RAE та FlagBag

Це може свідчити про варіабельність в затримках або про вплив інших неврахованих факторів. Загальна тенденція графіка показує, що з вищими значеннями затримки FlagBag значення затримки RAE також збільшуються, особливо помітно це у діапазоні від 9 до 11 секунд для FlatBag.

Перевірка гіпотез

Група FlagBag мала нижчі значення (M = 9.01, SD = 0.99), ніж група NRAE (M = 9.89, SD = 1.12). Це дало p-значення < 0.001, що нижче за заданий рівень значущості 0.05. Результати t-тесту є значущими для цих даних, і нульову гіпотезу відхилено. Отже, вважається, що обидві вибірки належать до різних сукупностей. Нульову гіпотезу про те, що змінна $NRAE$ ($0.9\mu_{RAE}$) має менше або рівне значення порівняно зі змінною FlagBag, перевіряли за допомогою одностороннього тесту за t-критерієм Стьюдента для парних вибірок. Результат виявився статистично значущим ($t(62) = -8.27$, $p < 0.001$). Нульову гіпотезу відхилено. Розмір ефекту d дорівнює 1.04. Ефект d = 1.04 є великим.

Висновки

Розроблено та досліджено новий метод серіалізації даних FlagBag, орієнтований на зменшення затримки міжсервісної комунікації в розподілених системах. Основною метою дослідження було досягнення зниження затримки передачі даних щонайменше на 10 % порівняно з Apache Avro, який було визначено, як найбільш продуктивний формат серіалізації в попередніх дослідженнях. Ця мета була досягнута та статистично перевірена.

Метод FlagBag демонструє середнє зниження затримки на 14- 22 % порівняно з Apache Avro залежно від умов навантаження та структури даних. Це перевищує мінімальний поріг, встановлений як ціль дослідження. Обсяг переданих даних у методі FlagBag скорочено на 15 % у середньому шляхом оптимізації структури даних, що також позитивно впливає на продуктивність системи. Практична реалізація методу підтверджує його сумісність із сучасними мікросервісними архітектурами без значних змін у процесах інтеграції.

Отримані результати дають змогу рекомендувати метод FlagBag для використання в системах, які потребують низької затримки передачі даних та економного використання ресурсів, зокрема в системах реального часу, IoT-системах та фінансових платформах. Зменшення затримки сприяє підвищенню швидкодії систем, а також забезпечує зниження витрат на мережеві ресурси.

Подальші напрями досліджень можуть передбачати оптимізацію методу FlagBag для роботи з іншими типами даних, вдосконалення процесу зворотної сумісності при додаванні та видаленні значень в нових версіях схеми повідомлення, дослідження впливу методу на енергоспоживання у пристроях з обмеженими ресурсами (IoT).

Подяки

У статті використано матеріали та результати, які отримали автори під час науково-дослідної роботи “Інтелектуальні методи та інструменти проєктування модульних автономних кіберфізичних систем”, НДР 0124U002340 від 09.03.2024, яка проводиться на кафедрі ЕОМ Інституту комп’ютерних технологій, автоматики та метрології національного університету “Львівська політехніка” у 2024–2028 роках.

Список літератури

1. *Hwang S. H., Kim K. M., Kim S., Kwak J. W. Lossless data compression for time-series sensor data based on dynamic bit packing. Sensors. 2023. 23(20). 8575. https://doi.org/10.3390/s23208575.*

2. *Abdelwahed M. F. A hybrid method for data compression and encryption based on bit packing, 128-based numerals, and bitmap manipulations: application to seismic data. Multimedia Tools and Applications. 2020. 79(31). 22705-22726. 10.1007/s11042-020-09082-3.*

3. *Clark M. A., Howarth D., Tu J., Wagner M., Weinberg E. Maximizing the Bang Per Bit (No. arXiv: 2302.09224, p. 338). 10.22323/1.430.0338. 2023.*

4. *Prammer Martin, Jignesh M. Patel. Rethinking the Encoding of Integers for Scans on Skewed Data. Proceedings of the ACM on Management of Data. 2023. 1.4. 1- 27. https://doi.org/10.1145/3626751.*

5. *Jiang H., Liu C., Paparrizos J., Chien A. A., Ma J., Elmore A. J. Good to the last bit: Data-driven encoding with codecdb. In Proceedings of the 2021 International Conference on Management of Data. 2021, June. 843- 856. https://doi.org/10.1145/3448016.3457283.*

6. *Sung S., Ko S. K., Han Y. S. Smaller Representation of Compiled Regular Expressions. In International Conference on Implementation and Application of Automata. 2023, August. Pp. 290- 301. Cham: Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-40247-0_22.*

7. *Wang H., Song S. Frequency domain data encoding in apache iotdb. Proceedings of the VLDB Endowment. 2022. 16(2). 282- 290. https://doi.org/10.14778/3565816.3565829.*

8. *Idir Y., Moumen I., Abouchabaka J., Rafalia N. (2024). Enhancing IoT Data Integrity and Effectiveness through hybrid Compression Method: A Step Towards Energy Efficiency. In E3S Web of Conferences. Vol. 477. P. 00042. EDP Sciences. https://doi.org/10.1051/e3sconf/202447700042.*

9. *Chen J. A., Sung H. H., Shen X., Tallent N., Barker K., Li A. Accelerating matrix-centric graph processing on GPUs through bit-level optimizations. Journal of Parallel and Distributed Computing. 2023. 177. 53- 67. https://doi.org/10.1016/j.jpdc.2023.02.013.*

10. *Li Y., Lu J., Chandramouli B. Selection Pushdown in Column Stores Using Bit Manipulation Instructions. Proceedings of the ACM on Management of Data. 2023. 1(2). 1- 26. https://doi.org/10.1145/3589323.*

11. *Bammes B., Spilman M. High-Temporal Resolution Event Streaming for Electron Counting. 2024. https://doi.org/10.1093/mam/ozae044.845.*

12. *Sakr F., Berta R., Doyle J., Younes H., De Gloria A., Bellotti F. Memory Efficient Binary Convolutional Neural Networks on Microcontrollers. In 2022 IEEE International Conference on Edge Computing and Communications (EDGE). 2022, July. Pp. 169- 177. 10.1109/EDGE55608.2022.0003289p.*

13. *Maltsev E. Enhancing Inter-Service Communication Through Multi-Baseline Delta Encoding, 2024 IEEE 17th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), Lviv, Ukraine, 2024. 10.1109/TCSET64720.2024.10755788.*

14. *Maltsev E., Muliarevych O. Beyond Json: Evaluating Serialization Formats for Space-Efficient Communication. ACPS. 2024. 9(1). 9- 15. https://doi.org/10.23939/acps2024.01.009.*

OPTIMIZING COMMUNICATION IN HIGHLY LOADED SYSTEMS USING FLAGBAG METHOD

E. Y. Maltsev, O. V. Muliarevych

Lviv Polytechnic National University,
Department of Electronic Computing Machines

© Maltsev E. Y., Muliarevych O. V., 2025

The paper presents a new method for optimizing data serialization for inter-service communication in distributed systems, called FlagBag. The proposed method aims to reduce the latency of data transmission between services by implementing an efficient data structure organization and serialization algorithm. The study was conducted using Apache Avro as a baseline format for comparison. Experimental results show that FlagBag reduces the average latency of data transmission between services by 18 % compared to unmodified Avro, and the amount of transmitted data is reduced by 15 % in some cases. In addition, the proposed method demonstrates stable performance when increasing the message size to 10 KB, providing an average 15 % advantage in transmission time in such a scenario. The paper also considers aspects of integrating FlagBag into existing microservice architectures, including the potential to reduce operational costs for supporting services in highly loaded systems. The performance tests confirmed the advantages of the method under real-world workload conditions, making FlagBag a promising solution for solving tasks with high requirements for speed and efficiency of inter-service communication. The proposed approach is universal and can be adapted for other serialization formats, providing performance improvements in a wide range of applications.

Keywords: Data communication, Encoding, Information exchange, Protocols, Performance evaluation.