НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.021

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2025 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою
«Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Модифікований метод та програмне забезпечення для контенто-залежної фрагментації даних»

Виконав:
Студент ІІ курсу, групи КП-41мп
Калашников-Травін Владислав Володимирович _____

Керівник:
Старший викладач кафедри ПЗКС, к.т.н.,
Хіцко Яна Володимирівна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри СПСКС, к.т.н., доцент,
Боярінова Юлія Євгенівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___»_____2024 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Калашникову-Травіну Владиславу Володимировичу

1. Тема дисертації «Модифікований метод та програмне забезпечення для контенто-залежної фрагментації даних», науковий керівник дисертації Хіцко Яна Володимирівна, к.т.н., старший викладач кафедри ПЗКС, затверджені наказом №4836-С по університету від 6 листопада 2025 р.

2. Термін подання студентом дисертації 19 грудня 2025 р.

3. Об'єкт дослідження: процес дедублікації даних.

4. Предмет дослідження: способи та методи контенто-залежної фрагментації даних.

5. Перелік завдань, які потрібно розробити:
   - провести аналіз відомих методів контенто-залежної фрагментації даних;
   - порівняти характеристики фрагментації та дедублікації розглянутих методів контенто-залежної фрагментації даних, виокремити їх переваги та недоліки;
   - розробити модифікацію одного з вже відомих методів контенто-залежної фрагментації даних з поліпшеними характеристиками продуктивності та/або дедублікації;
   - створити програмну реалізацію запропонованого методу та переконатися в коректності її роботи;
   - реалізувати програмне забезпечення для емпіричного порівняння та аналізу методів контенто-залежної фрагментації даних;
   - провести порівняльний аналіз розглянутих та запропонованого методу, використовуючи результати отримані реалізованим програмним забезпеченням;
   - розробити бізнес-модель.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
   - блок-схема базового методу Fast CDC;
   - блок-схема модифікованого методу Twin CDC;
   - блок-схема механізму збору метрик продуктивності;
   - діаграма компонентів розробленого програмного забезпечення;
   - гістограми емпіричних результатів порівняння розглянутих та модифікованого методів.

7. Орієнтовний перелік публікацій:
   - Тези доповіді “Модифікований метод та програмне забезпечення для контенто-залежної фрагментації даних”

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент кафедри ПЗКС		

9. Дата видачі завдання «04» жовтня 2025 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.10.2024	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.12.2024	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.02.2025	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	05.04.2025	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.05.2025	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	15.06.2025	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2025	05.11.2025	
8.	Оформлення текстової і графічної частини магістерської дисертації	04.12.2025	

Студент Владислав КАЛАШНИКОВ-ТРАВІН

Науковий керівник Яна ХІЦКО

РЕФЕРАТ

Актуальність теми. У сучасних розподілених та хмарних сховищах даних, системах резервного копіювання та синхронізації обсяги інформації зростають експоненційно, при цьому значна частка нових даних утворюється шляхом модифікації вже наявних версій файлів, віртуальних машин або баз даних. Це призводить до накопичення великої кількості логічно тотожних або майже тотожних фрагментів, що суттєво підвищує витрати на дисковий простір та пропускну здатність мережі. Одним із ключових механізмів зменшення цих витрат є дедублікація даних, ефективність якої критично залежить від якості роботи методів фрагментації, насамперед від методів контенто-залежної фрагментації даних.

Класичні методи демонструють суперечливий компроміс між пропускною здатністю, коефіцієнтом дедублікації та розподілом розмірів фрагментів, зокрема високу девіацію розмірів фрагментів даних, тоді як новітні модифікації часто вимагають тонкого налаштування під конкретні сценарії та не забезпечують оптимальних характеристик на довільних наборах даних.

Отже, дослідження методів контенто-залежної фрагментації даних є актуальним у контексті сучасних обсягів новоутворених даних, а вдосконалення цих методів прямо впливатиме на зменшення витрат на зберігання та передавання великих обсягів інформації, що зумовлює високу актуальність обраної тематики для сучасних систем збереження даних.

Об’єктом дослідження є процес дедублікації даних.

Предметом дослідження є способи та методи контенто-залежної фрагментації даних.

Метою роботи є збільшення продуктивності фрагментації даних, підвищення коефіцієнту дедублікації даних, а також запобігання утворенню аномально малих та великих фрагментів даних шляхом модифікації одного з вже відомих методів контенто-залежної фрагментації даних.

Методами дослідження є теоретичний та емпіричний аналіз відомих методів контенто-залежної фрагментації даних.

Наукова новизна. Вперше було розроблено модифікований метод контенто-залежної фрагментації даних Twin CDC, який завдяки двонаправленому пошуку меж фрагментів, використанню двох окремих Gear-таблиць та встановленню альтернативних меж фрагментів забезпечує підвищення пропускної здатності фрагментації даних від 2% до 39%, збільшення пропускної здатності дедублікації від 3% до 46 % та зростання коефіцієнта дедублікації даних на 3 % відносно вже відомих методів фрагментації даних.

Практична значущість. За результатами даного дослідження було виявлено, що запропонована модифікація має кращі кількісні характеристики продуктивності та дедублікації, що робить дану модифікацію придатною для використання у виробничих середовищах, зокрема у системах резервного копіювання, медіа-платформах, а також в інфраструктурі постачальників хмарного постійного сховища.

Апробація роботи. Основні положення і результати роботи були представлені у роботі «Модифікований метод та програмне забезпечення для контенто-залежної фрагментації даних» та обговорювалися на науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг 2025», що проходила з 19 по 21 листопада 2025 року у м. Київ.

Структура та обсяг роботи. Магістерська дисертація складається зі вступу, п’яти розділів, висновків та додатків.

У вступі було наведено загальну характеристику роботи, проаналізовано сучасний стан розв’язання відповідної науково-практичної проблеми, детально обґрунтовано актуальність обраного напрямку досліджень, а також сформульовано мету та основні завдання роботи.

У першому розділі було розглянуто процес дедублікації даних, його основні етапи з фокусом на перший та найголовніший етап – контенто-залежну фрагментацію даних. Було виконано порівняльний аналіз

вже відомих методів фрагментації даних, виділено їх окремі та спільні переваги та недоліки. За результатами аналізу було виявлено необхідність у створенні нового або модифікації вже відомого методу фрагментації з метою усунення виявлених недоліків відомих методів.

У другому розділі було запропоновано модифікований метод Twin CDC, що базується на вже відомому методі Fast CDC. Було описано три його характерні відмінності, завдяки яким має бути досягнуто мети даної роботи.

У третьому розділі було сформульовано вимоги до розроблюваного програмного продукту, описано його архітектуру та складові, а також висвітлено особливості реалізації модифікованого методу Twin CDC та супутнього програмного забезпечення для порівняльного аналізу методів контенто-залежної фрагментації.

У четвертому розділі, з використанням розробленого програмного забезпечення було проведено емпіричний порівняльний аналіз відомих та модифікованого методів контенто-залежної фрагментації, а також наведено характеристику кожного з отриманих результатів. Крім того, було висвітлено подальші напрямки роботи, що потенційно можуть дедалі покращити метрики продуктивності фрагментації та дедублікації даних.

У п’ятому розділі було детально описано бізнес-модель стартап-проєкту *ChunkIt,* характеризовано основні вирішувані проблеми, зацікавлені сторони, запропоноване інноваційне рішення, а також структуровано потенційні джерела доходу та прогнозовані витрати.

У висновках підсумовано отримані результати дослідження та виділено його ключові положення та досягнення роботи.

Дана робота виконана на 98 аркушах, містить 18 рисунків, 5 таблиць, а також список використаних літературних джерел з 36 найменувань.

Ключові слова: інженерія програмного забезпечення, фрагмент, фрагментація, дедублікація, CDC.

ABSTRACT

Relevance of the topic. In modern distributed and cloud storage systems, backup and data synchronization systems, data volumes are growing exponentially, while a significant portion of new data is generated by modifying existing versions of files, virtual machines or databases. This leads to the accumulation of a large number of logically identical or nearly identical fragments, which significantly increases the costs of disk space and network bandwidth. One of the key mechanisms for reducing these costs is data deduplication, whose effectiveness entirely depends on the quality of chunking methods, primarily content-defined chunking methods. Classic methods exhibit a dubious trade-off between throughput, deduplication ratio and chunk-size distribution, in particular a high deviation of chunk sizes, whereas recent modifications often require careful tuning for specific scenarios and do not provide optimal characteristics on arbitrary datasets.

Therefore, the study of content-defined chunking methods is relevant in the context of modern volumes of newly generated data, and the improvement of these methods will directly affect the reduction of storage and transmission costs for large amounts of information, which determines the high relevance of the chosen topic for modern data storage systems, specifically for backup

The **object** of the research is the process of data deduplication.

The **subject** of the research is the techniques and methods of content-defined chunking.

The **aim** of the thesis is to increase data chunking performance, improve the deduplication ratio, and prevent the formation of abnormally small and large data fragments by modifying one of the existing content-defined chunking methods.

The **research methods** are theoretical and empirical analysis of the known content-defined chunking methods.

Scientific novelty. For the first time, a modified method of content-defined chunking, Twin CDC, was developed, which, thanks to bidirectional search for

chunks boundaries, the use of two separate Gear tables, and the establishment of alternative fragment boundaries, provides an increase in data chunking throughput from 2% to 39%, increases deduplication throughput by 3% to 46%, and increases the data deduplication ratio by 3% compared to well-known content-defined chunking methods.

Practical significance. The results of the study have shown that the proposed modification exhibits better quantitative characteristics of performance and deduplication, which makes it suitable for use in production environments, particularly in backup systems, media platforms, and the infrastructure of cloud storage providers.

Approbation of the work. The main statements and results of the thesis were presented in the paper “Modified method and software for content-defined chunking of data” and discussed at the scientific conference of master’s and PhD students “Applied Mathematics and Computing 2025”, held on 19–21 November 2025 in Kyiv.

Structure and scope of the thesis. The master’s thesis consists of an introduction, five chapters, conclusions and appendices.

The introduction provides a general description of the work, analyses the current state of the corresponding scientific and practical problem, substantiates in detail the relevance of the chosen research direction, and formulates the aim and main objectives of the thesis.

The first chapter considers the process of data deduplication and its main stages with a focus on the first and key stage of content-defined chunking. A comparative analysis of the existing content-defined chunking methods is carried out, and their individual and common advantages and disadvantages are identified. Based on this analysis, the need to create a new method or to modify an existing chunking method to eliminate the identified drawbacks is established.

The second chapter proposes the modified method Twin CDC, which is based on the well-known Fast CDC method. Three characteristic differences of

the method are described, owing to which the aim of the thesis is expected to be achieved.

The third chapter formulates the requirements for the software system being developed, describes its architecture and components, and presents the implementation details of the modified Twin CDC method and the accompanying software for comparative analysis of content-defined chunking methods.

In the fourth chapter, using the developed software, an empirical comparative analysis of the known and the modified content-defined chunking methods is performed, and each obtained result is characterized. In addition, further research directions are outlined, which can potentially lead to further improvements in chunking and deduplication performance metrics.

The fifth chapter provides a detailed description of the business model of the *ChunkIt* startup project, characterizes the primary stakeholders, describes the main problems to be solved, presents the proposed innovative solution, and structures potential revenue streams and projected costs.

The conclusions summarize the obtained research results and highlight its key contributions and achievements.

The thesis is presented on 98 pages, contains 18 figures, 5 tables, and a list of 36 references.

Keywords: software engineering, chunk, chunking, deduplication, CDC.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

FSC (Fixed-Size Chunking) – фрагментація даних фіксованої довжини, за якої вхідний потік даних перетворюється на послідовність фрагментів однакового розміру за фіксованою константою розміру фрагментів.

CDC (Content-Defined Chunking) – контенто-залежна фрагментація даних, за якої вхідний потік даних перетворюється на послідовність фрагментів даних різного розміру за ознаками вмісту цього потоку даних.

IO (Input/Output) – операції вводу/виводу, такі як читання/запис на диск, отримання мережевих пакетів тощо.

Бенчмарк – стандартизований тест або набір тестів, призначений для вимірювання та порівняння продуктивності апаратних чи програмних систем за заздалегідь визначеним набором метрик.

CIL (Common Intermediate Language) – проміжна незалежна від платформи мова низького рівня, у яку компілюються програми платформи .NET під час компіляції.

Big/little endianness – порядок збереження старших та молодших байтів чисел у пам’яті.

JIT (Just-In-Time) – механізм виконання програм, за якого код проміжної мови CIL перетворюється на машинний код безпосередньо під час виконання програми.

GC (Garbage Collector) – механізм автоматичного керування та звільнення пам’яті, що є компонентом середовища виконання програм платформи .NET.

Tiered compilation – багаторівнева JIT-компіляція, що включає різні рівні оптимізації програмного коду.

Dynamic PGO (Dynamic Profile-Guided Optimization) – механізм оптимізації коду, за якого середовище виконання збирає профіль реального виконання програми під час її роботи і на основі цих даних перекомпілює вимогливі ділянки коду з вищим рівнем низькорівневої оптимізації.

Nice level – числовий параметр, який задає відносний пріоритет процесу для планувальника ядра.

Thread (потік виконання) – найменша одиниця планування роботи програми, послідовність інструкцій якої виконується незалежно від інших потоків.

SLA (Service Level Agreement) – формалізована угода між постачальником послуги та її споживачем, яка визначає гарантовані рівні якості сервісу, наприклад, доступність, допустимі затримки, час реакції та відновлення.

CI/CD (Continuous Integration / Continuous Delivery) – сукупність практик і автоматизованих процесів у розробці ПЗ, що забезпечують безперервну інтеграцію, тобто часте об’єднання змін у спільну гілку з автоматичними збірками та тестами, та безперервне постачання, тобто автоматизована підготовка та розгортання у виробничому середовищі.

API (Application Programming Interface) – формально визначений інтерфейс взаємодії між програмними компонентами/системами, що задає доступні операції, формати даних, правила виклику та обміну повідомленнями.

B2B (Business to Business) – модель взаємодії, у якій продукт або послуга надається одним бізнесом іншому бізнесу.

B2C (Business to Consumer) – модель взаємодії, у якій продукт або послуга надається бізнесом кінцевому споживачу.

ВСТУП

У сучасному діджиталізованому середовищі обсяги щоденно створених, скопійованих та переданих даних демонструють експоненційне зростання [1]. Так, наприклад, вагомий внесок у цей приріст роблять платформи масового розповсюдження мультимедійного контенту, включаючи YouTube, TikTok та Instagram, де щодня публікуються колосальні об’єми відео- та аудіо-матеріалів [2]. Основне інфраструктурне навантаження з попередньої обробки, зберігання та обслуговування покладається на провайдерів хмарних сервісів, зокрема на сервіси довгострокового, високодоступного та масштабованого збереження.

За таких умов критичними стають методи ефективного використання дискових і мережевих ресурсів, оскільки без систематичного усунення надлишковості витрати на сховище та пропускну здатність зростають швидше, ніж корисний інформаційний вміст [3].

Одним із підходів до оптимізації використання дискових та мережевих ресурсів є дедублікація даних – процес усунення надлишкових даних шляхом виявлення та вилучення повторюваних фрагментів. Процес дедублікації складається з кількох етапів, серед яких ключовим етапом є контенто-залежна фрагментація даних, за якого вхідний потік даних перетворюється у послідовність фрагментів даних, далі серед яких відбуватиметься пошук та вилучення повторюваних входжень. Саме від методу фрагментації даних залежить продуктивність та якісь процесу дедублікації, що, в свою чергу, прямо впливає на економічність підтримки апаратної інфраструктури.

Нині відомо цілу низку методів контенто-залежної фрагментації даних. Серед таких методів є класичні підходи, що мають задовільний рівень дедублікації, однак демонструють низьку продуктивність та утворюють надмірну кількість аномально малих або великих фрагментів. Новітні ж методи характеризуються вищою продуктивністю та кращим розподілом розмірів фрагментів даних, проте такі методи часто вимагають

тонкого налаштування під конкретний тип даних, а за обробки монотонних чи абсолютно випадкових потоків даних ці методи також втрачають продуктивність та якість дедублікації.

З метою усунення наведених недоліків відомих методів контенто-залежної фрагментації даних, у даному досліджені було детально розглянуто характеристики відомих методів, проаналізовано їх переваги та недоліки та на основі проведеного аналізу було запропоновано модифікований метод контенто-залежної фрагментації даних, що завдяки трьом ключовим відмінностям демонструє вищу продуктивність фрагментації даних, вищий коефіцієнт дедублікації даних, а також усуває проблему утворення аномальних фрагментів на довільних наборах даних.

1. АНАЛІЗ ВІДОМИХ МЕТОДІВ ФРАГМЕНТАЦІЇ ДАНИХ

1.1. Задача дедублікації даних

У типових робочих навантаженнях значна частка новоутворених даних не є унікальною, оскільки більшість таких даних створюється шляхом редагування вже існуючих даних [4]. До таких даних можна віднести версіоновані документи, репозиторії програмного коду, регулярні резервні копії файлів, баз даних та віртуальних машин. У таких сценаріях формуються множинні копії логічно тотожних або майже тотожних масивів, де відмінності зводяться до локальних вставок, зсувів та незначних правок. На рис. 1 зображена надлишковість збереження двох версій одного файлу, де між першою та другою версіями одного файлу було змінено фрагменти $D \rightarrow D'$ та $E \rightarrow E'$, в той час як фрагменти *A*, *B* та *C* не були змінені.

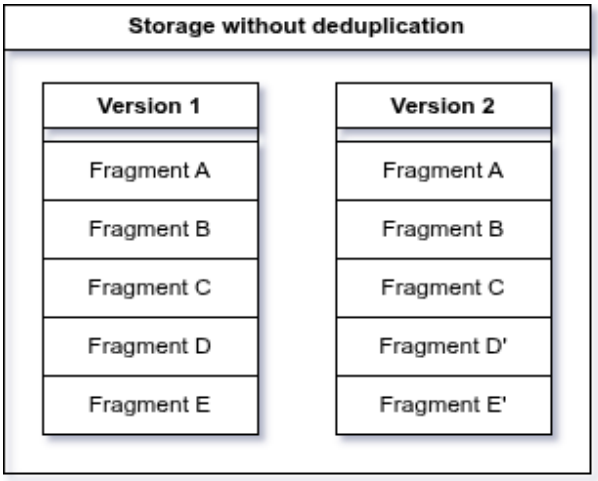


Рис. 1. Надлишковість зберігання даних двох версій одного файлу без використання дедублікації

Не зважаючи те, що версії відрізняються лише двома фрагментами, у сховищі зберігаються дві окремі копії файлу, а загальний використаний дисковий простір SS_p без дедублікації даних розраховується згідно:

$$SS_p = 2 * A + 2 * B + 2 * C + D + E + D' + E'. \quad (1)$$

Дедублікація усуває наведену надмірність шляхом зберігання лише одного екземпляру кожного повторюваного фрагмента, а всі наступні

входження представляються посиланнями на вже наявний екземпляр [5]. На рис. 2 зображено зберігання двох версій одного файлу з використанням дедублікації.

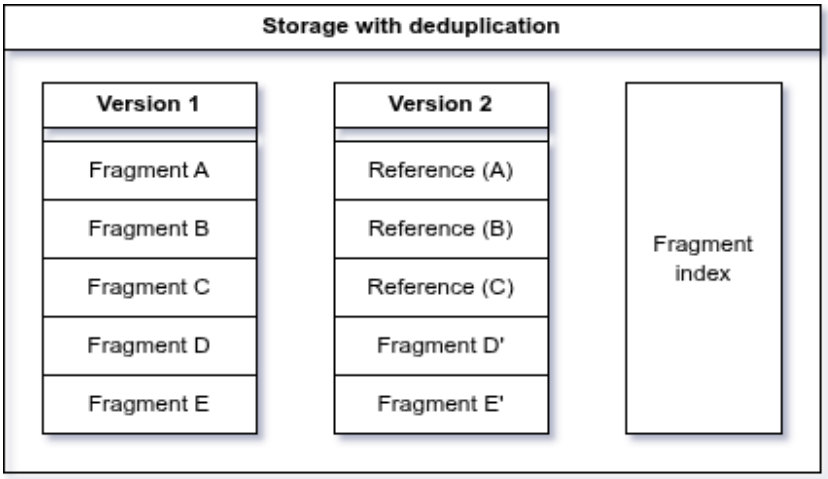


Рис. 2. Відсутність надлишковості зберігання даних двох версій одного файлу з використанням дедублікації

Дубліковані фрагменти A, B та C тепер зберігаються у єдиному екземплярі, а їх копії були замінені на відповідні посилання. Такі посилання зазвичай зберігаються у індексі або так званому “рецепті” [6], згідно якого можна відтворити файл, замінюючи посилання на фрагменти. Обсяг дедублікованого дискового простору SS_d та індексу SS_{idx} розраховуються як:

$$SS_d = A + B + C + D + E + D' + E' + SS_{idx}. \quad (2)$$

$$SS_{idx} = \sum_{i=0}^{N-1} meta(f_i) \quad (3)$$

Слід зауважити, що SS_i значно менший за загальний обсяг дедублікованих даних. Такий підхід водночас скорочує потребу у необхідному дисковому просторі та зменшує мережеві витрати на передачу даних між пристроями. Так, наприклад, під час резервного копіювання чи синхронізації даних передаються лише нові або змінені фрагменти. Відповідно, зі зменшенням переданих та збережених даних зменшується і вартість утримання та обслуговування апаратних ресурсів.

Основною характеристикою дедублікації даних є коефіцієнт дедублікації D, що розраховується відповідно до (4), при чому чим ближчий

D до правої межі, тим кращою характеризується дедублікація. Очевидно, що отримати значення *D = 1* неможливо, оскільки це фактично означатиме, що певний об'єм даних SS_p можливо стиснути до абсолютного нуля, що суперечить теоремі Шеннона про стиснення даних [7].

$$D = \frac{SS_p - SS_d}{SS_p} \in [0; 1) \quad (4)$$

Іноді при розрахунку коефіцієнту *D* нехтують розміром індексу SS_i, оскільки, зазвичай, зберігають такий індекс окремо від даних, що фрагментуються [8]. В такому випадку, коефіцієнт *D* розраховується згідно до:

$$D = \frac{SS_p - (SS_d - SS_{idx})}{SS_p} \in [0; 1). \quad (5)$$

Конвеєр процесу дедублікації даних, що зображений на рис. 3, складається з чотирьох основних етапів, результатом виконання яких є набір унікальних фрагментів обраного файлу та індекс посилань на фрагменти [9].

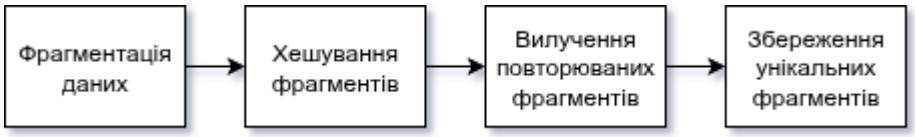


Рис. 3. Конвеєр процесу дедублікації даних

1.1.1. Фрагментація даних

Фрагментація даних є першим і визначальним етапом дедублікації, оскільки на цьому кроці суцільний байтовий потік перетворюється на послідовність фрагментів, серед яких і виконується пошук повторів. Встановлення меж фрагментів визначається предикатом встановлення межі *P*, форма якого залежить від обраного методу фрагментації даних.

Фрагмент – це атомарна неперервна послідовність байтів виділена з повного набору даних, що може бути розглянута як самостійна одиниця даних, яка є придатною для зберігання та адресації. Як показано у (6), впорядкована послідовність фрагментів складає повний вихідний файл.

$$F = \sum_{k=0}^{L} f(i_k, o_k, l_k) \quad (6)$$

Згідно (6), кожен фрагмент f_k визначається трьома основними властивостями, що є незмінними та дозволяють однозначно адресувати вміст цього обраного фрагменту.

Порядковий номер i_k. Фрагмент f_k має власний порядковий номер, що дозволяє визначити порядок входження фрагменту у загальній послідовності фрагментів обраного файлу. Слід зауважити, що порядковий номер не виступає унікальним ідентифікатором фрагменту, а лише вказує на порядок входження фрагменту у загальній послідовності, оскільки унікальність фрагменту визначається виключно його вмістом.

Глобальне зміщення o_k. Фрагмент f_k має однозначно визначене місцезнаходження у вихідному потоці байтів. Початок фрагменту задається глобальним зміщенням o_k – позицією першого (в програмуванні – нульового) байта від початку вихідного файла. Глобальне зміщення виступає внутрішнім просторовим індексом, що фіксує позицію початку кожного фрагмента та гарантує коректність операцій відтворення та адресації.

Довжина фрагменту l_k. Фрагмент f_k має певну ненульову довжину l_k. Разом з глобальним зміщенням o_k, пара (o_k, l_k) формує напів відкритий інтервал $d_k = [o_k, o_k + l_k)$, який повністю визначає інтервал у вихідному потоці байтів, що охоплюється обраним фрагментом f_k.

Нехай F – це скінченна впорядкована послідовність байтів вихідного файлу довжиною L. Тоді впорядкований набір фрагментів $M = \{ f_k \}$ довжиною N, де фрагмент $f_k = (i_k, o_k, l_k)$ та відповідний інтервал фрагменту $d_k = [o_k, o_k + l_k)$, коректно представляє вихідний файл F тоді і тільки тоді, коли виконуються наступні три умови.

Умова 1. Відсутність перетинів інтервалів. Будь-які два обрані інтервали фрагментів f_a та f_b не мають перетинатися (мати спільних байтів) у своїх послідовностях, відповідно до:

$$\forall a, b \in [0, N), a \neq b: d_a \cap d_b = \emptyset. \quad (7)$$

Умова 2. Відсутність розривів між інтервалами. Кожен наступний інтервал f_{k+1} має мати таке глобальне зміщення o_{k+1}, що $o_{k+1} = o_k + l_k$, таким чином утворюючи повне покриття вихідної послідовності байтів, відповідно до:

$$\cup_{k=0}^{N-1} d_k = [0, L). \quad (8)$$

Умова 3. Збереження довжини. Сума розмірів всіх фрагментів має бути рівною розміру вихідного файлу, відповідно до:

$$\sum_{k=0}^{N-1} l_k = L. \quad (9)$$

Вибір цільового розміру фрагменту визначає подальший баланс між розміром індексу та вірогідністю знайти дубліковані фрагменти [8]. Щоменші фрагменти, тим більша ймовірність знайти відповідні дублікати, але й з тим зростає розмір індексу, що зберігає інформацію про фрагменти та їх розташування. З іншої сторони, чим більші фрагменти, тим менший розмір індексу, але й менша ймовірність повторного використання фрагментів. Зазвичай, незалежно від методу фрагментації даних та відповідного предикату P, цільовий розмір фрагментів налаштовується трьома значеннями $S_{conf} = (S_{min}, S_{avg}, S_{max})$, що робить довжини фрагментів більш керованими та передбачуваними [10].

Мінімальний розмір фрагменту S_{min}. Суворе обмеження на встановлення меж фрагментів до досягнення заданої мінімальної довжини запобігає утворенню занадто малих фрагментів, що може призвести до надмірних розмірів індексу фрагментів. Єдиним виключенням з цього обмеження є випадок, коли формується останній фрагмент f_{N-1} з залишку потоку байтів, тобто $l_{N-1} < S_{min}$.

Середній бажаний розмір фрагменту S_{avg}. Це значення задає очікувану середню довжину фрагментів, і, відповідно, очікувану кількість утворених фрагментів $E = L / S_{avg}$, при чому $N \to E$.

Максимальний розмір фрагменту S_{max}. Накладення суворого обмеження на встановлення меж фрагментів після S_{max} необхідне з метою

уникнення фрагментів надмірної довжини за умови, що предикату P не вдалося визначити межу до досягнення S_{max}.

Очевидно, що для будь-яких обраних значень параметрів цільового розміру фрагментів S_{conf} має виконуватися нерівність:

$$S_{min} < S_{avg} < S_{max}. \quad (10)$$

Проте, практична якість фрагментації даних визначається не лише кількістю знайдених повторів.

Стійкість меж. Будь-який обраний метод фрагментації даних характеризується здатністю зберігати межі фрагментів даних за умови внесення локальних модифікацій у вихідний файл, такі як вставки на початку, всередині або в кінці, вилучення підрядків, поодинокі заміни тощо.

Нехай у вихідний файл було зроблено вставку довжиною M, тоді така вставка спричиняє коливання меж в околі W, де W – ефективна ширина контексту, за яким визначається межа. В той же час межі фрагментів поза околом W вирівнюються з попередніми межами, при цьому вже маючи додатковий зсув на M байтів, як зображено на рис. 4. Таке вирівнювання називають ресинхронізацією меж фрагментів даних [11], саме воно забезпечує локальність наслідків правок. За відсутності ресинхронізації меж фрагментів даних навіть мінімальна вставка спричиняє каскадний зсув усіх подальших меж, що руйнує збіги між версіями та різко знижує коефіцієнт дедублікації D.

Стійкість до локальних правок та видалень, а також ресинхронізація меж фрагментів даних є ключовими показниками якості фрагментації, оскільки прямо визначають частку повторно використаних фрагментів між версіями та, відповідно, якість дедублікації даних, а отже і економія вартості придбання та обслуговування постійного сховища та пропускної здатності мережевих каналів.

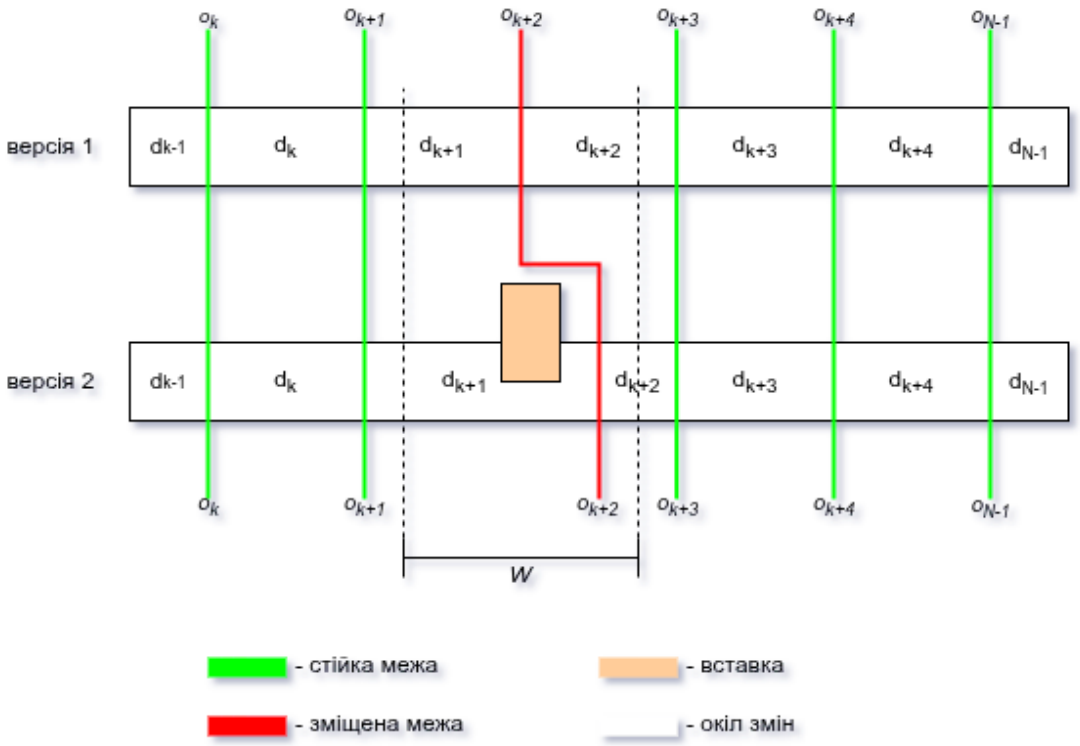


Рис. 4. Ресинхронізація меж між двома версіями файлу

Пропускна здатність фрагментації даних. Емпірична пропускна здатність фрагментації даних T_f розраховується за (11), де t – це час фрагментації обраного файлу обсягом SS_p. Варто відзначити, що практично всі методи фрагментації даних мають порівняно низьку пропускну здатність на потоках випадкових байтів у порівнянні зі структурованими потоками.

$$T_f = \frac{SS_p}{t} \tag{11}$$

Пропускна здатність дедублікації даних. Зведена міра пропускної здатності дедублікації даних T_d розраховується за (12), визначає обсяг дедублікованих даних за одиницю часу, демонструючи при цьому ефективність вилучення дублікованих фрагментів відносно витраченого часу на визначення меж цих фрагментів.

$$T_d = \frac{SS_p - SS_d}{t} \tag{12}$$

Розподіл розмірів фрагментів. Розподіл розмірів фрагментів, що характеризує набір властивостей фрагментів { l_k }, буває трьох типів: вироджений розподіл (дельта-розподіл навколо сталої величини),

геометричний розподіл (кожна позиція має приблизно сталу ймовірність стати межею), а також нормалізований розподіл, який штучно звужують за допомогою значень S_{conf}. З точки зору дедублікації даних, оптимальним розподілом вважається унімодальний низькодисперсний розподіл із центром біля S_{avg}, без великої кількості фрагментів близьких за довжиною до S_{min} або S_{max}. Такий розподіл дає стабільну щільність меж та краще розрізняє повторювані фрагменти даних [7].

Геометричний коефіцієнт якості фрагментації даних. Очевидно, що зі зменшенням цільового розміру фрагментів S_{conf} має зрости кількість дублікатів фрагментів даних, адже що менший вміст кожного фрагменту, тим менша послідовність байтів, що має співпасти між будь-якими двома обраними фрагментами даних для усунення надлишковості одного з них. З іншого боку, чим більша кількість утворених фрагментів даних, тим більший розмір індексу та більші накладні витрати на відновлення файлу з фрагментів [13]. Враховуючи обидві наведені особливості, для досягнення оптимального процесу фрагментації даних необхідно підтримувати баланс розміру фрагментів та коефіцієнту дедублікації даних. З метою отримання кількісної оцінки такого балансу пропонується ввести геометричний коефіцієнт якості фрагментації даних. Даний коефіцієнт має кількісно відобразити отриманий баланс на основі двох значень: коефіцієнт дедублікації та девіація від цільового середнього розміру фрагментів. Нехай маємо значення найбільших можливих девіацій розмірів фрагментів U_{left} та U_{right} відносно S_{min} та S_{max} відповідно, які можна розрахувати за (13) та (14). Тоді, коефіцієнт девіації фрагментів даних V розраховується за (15). Маючи розраховані значення D та V, геометричний коефіцієнт якості фрагментації даних розраховується згідно (16).

$$U_{left} = S_{avg} - S_{min} \tag{13}$$

$$U_{right} = S_{max} - S_{avg} \tag{14}$$

$$V = 1 - \frac{1}{N}\sum_{k=0}^{N-1}\frac{|l_k - S_{avg}|}{u_k}, u_k = \begin{cases} U_{left}, & l_k \le S_{avg} \\ U_{right}, & l_k > S_{avg} \end{cases} \tag{15}$$

$$Q = \sqrt{\mathrm{D} \times \mathrm{V}} \in [0,1] \quad (16)$$

Ідеальним значенням коефіцієнту якості фрагментації даних є $Q = 1$. В такому випадку, коефіцієнт дедублікації $D = 1$, тобто об'єм збереженого дискового простору дорівнює об'єму оригінального файлу, а також коефіцієнт девіації фрагментів даних $V = 1$, тобто розмір всіх фрагментів даних дорівнює цільовому середньому розміру S_{avg}. Вочевидь досягти цього ідеального випадку неможливо, оскільки, як вже було показано раніше, $D \in [0; 1)$. Тим не менш, що ближче значення Q до 1, тим кращий баланс між розподілом розмірів утворених фрагментів та об'ємом збереженого дискового простору.

1.1.2. Хешування фрагментів

Перед безпосереднім вилученням однакових фрагментів, їх необхідно порівняти та віднайти такі що повторюються. Як вже було згадано, порядковий номер фрагментів не враховується при порівнянні, натомість унікальність фрагментів задається виключно контентом, тобто байтами, що кожен окремий фрагмент містить. Повторюваними вважаються такі фрагменти, що охоплюють ідентичний контент, незалежно від їх глобального зміщення o_k.

Найпростішим способом порівняти контент двох фрагментів є просте побайтне порівняння, оскільки цей спосіб не вимагає залучення додаткових алгоритмів. Однак, на противагу простоті постає проблема продуктивності, яка лінійно залежить від довжини порівнюваних фрагментів. Нехай маємо два фрагменти однакової довжини f_a та f_b, а також функцію побайтного порівняння фрагментів $Y(f_a, f_b)$, тоді алгоритмічну складність такого порівняння O_Y можна розрахувати за (17). Очевидно, що якщо фрагменти мають різні довжини $l_a \neq l_b$, то і порівнювати такі фрагменти не має потреби.

$$O_Y = O(l_a) \quad (17)$$

Така лінійна залежність погано масштабується на великих обсягах даних, тому що для пошуку дублікатів серед множини фрагментів

довжиною L побайтне порівняння вимагає завантаження у пам'ять та порівняння з кожним фрагментом-кандидатом, що дає алгоритмічну складність O_X, яка розраховується за (18).

$$O_X = O\left(L^2 \times S_{avg}\right) \tag{18}$$

Задля уникнення високої алгоритмічної складності побайтного порівняння, типово використовуються хеш-функції H, що виконують детерміністичне перетворення довільного потоку байтів у фіксований за довжиною відбиток даних h, що і використовується під час порівняння двох довільних фрагментів [13]. Висока алгоритмічна складність побайтного порівняння замінюється одноразовим викликом хеш-функції складністю $O(l_k)$, а всі подальші перевірки на наявність обраного фрагменту в індексі за відбитком складатиме $O(1)$.

Однак, не кожна хеш-функція придатна для порівняння вмісту фрагментів. Для коректної ідентифікації повторюваних фрагментів потрібен відбиток із криптографічними властивостями, насамперед стійкістю до колізій [14]. Нехай є два різних за вмістом фрагменти f_a та f_b. Згідно принципу Діріхле, серед множини b-бітних відбитків довжиною Q, будь-яка хеш-функція H має ненульову вірогідність P_c (15), з якою $H(f_a) = H(f_b)$. На практиці, поява такої колізії фактично означає пошкодження даних, оскільки замість вмісту фрагменту f_a може бути використаний вміст фрагменту f_b.

$$P_c \approx 1 - exp\left(-\frac{Q \times (Q-1)}{2^{b+1}}\right) \tag{19}$$

З метою унеможливлення появи колізій відбитків використовують криптографічно-стійкі хеш-функції, які мають $P_c(H) \rightarrow 0$ [5]. До таких функцій, наприклад, належать SHA-256, SHA-512 або MurmurHash3.

1.1.3. Вилучення повторюваних фрагментів

Вилучення повторюваних фрагментів є третім етапом конвеєра дедублікації даних та полягає у перетворенні послідовності фрагментів на дві сутності: множину унікальних фрагментів $U = \{ f_u \}$ з їхніми

відповідними відбитками h_u та повний індекс унікальних фрагментів, що містить список входжень кожного такого фрагменту з глобальним зміщенням o_u.

При створенні згаданого індексу використовуються відбитки утворених фрагментів. Якщо під час внесення фрагменту f_k в індекс виявляється, що фрагмент з відбитком $h_k = h_u$ вже існує, то в індекс додається лише глобальне зміщення o_k для фрагменту f_u. Інакше, в індекс додається новий фрагмент $f_u = f_k$ з допоки єдиним глобальним зміщенням o_k.

1.1.4. Збереження унікальних фрагментів

Після створення повного індексу фрагментів всі унікальні фрагменти записуються на диск, таким чином обраний файл зберігається без надмірності даних. Для подальших зверненнях до такого файлу, його необхідно реконструювати за допомогою його повного індексу фрагментів та збережених унікальних фрагментів. Власне, саме тому такі індекси називають “рецептами”, оскільки перед доступом до файлу його потрібно відтворити маючи всі необхідні “інгредієнти”, тобто фрагменти даних, та їх глобальні зміщення.

1.2. Аналіз відомих методів фрагментації даних

Двома нині відомими способами фрагментації даних є фрагментація даних з фіксованою довжиною фрагментів та контенто-залежна фрагментація даних. В той час, як у обох способів є власні сценарії використання, останній є більш витонченим способом фрагментації, завдяки якому і досягається дедублікація даних.

1.2.1. Фрагментація даних фіксованої довжини

Фрагментація даних фіксованої довжини (FSC) – це найпростіший спосіб фрагментації даних, в якому всі фрагменти, за винятком останнього, мають однакову довжину, тобто для будь-якого фрагменту f_k виконуються нерівності:

$$\forall k \in [0; N-1): l_k = S_{fixed}, \tag{20}$$

$$l_{N-1} = L \; mod \; S_{fixed}. \tag{21}$$

Раніше було введено поняття цільового розміру фрагментів S_{conf} та трійки його параметрів S_{conf} = (S_{min}, S_{avg}, S_{max}). У фрагментації даних з фіксованою довжиною, всі три параметри, а отже і цільовий розмір фрагментів S_{conf} рівний константі S_{fixed}.

Предикат фрагментації з фіксованою довжиною фрагменту P_{FSC}, як показано (22), також визначається за допомогою значення константи S_{fixed}.

$$P_{FSC} \coloneqq o_k \; mod \; S_{fixed} = 0 \tag{22}$$

Не дивлячись на простоту цього методу, він має один суттєвий недолік, а саме повна відсутність властивості стійкості меж в разі внесення локальних вставок або видалень. Навіть єдиний зсув на один байт спричиняє каскадне зміщення всіх подальших меж, що зображено на рис. 5.

За відсутності стійкості меж, даний метод непридатний до використання з метою дедублікації даних. Тим не менш, цей метод має свої випадки застосування, наприклад, у випадках де швидкість фрагментації даних відіграє ключову роль.

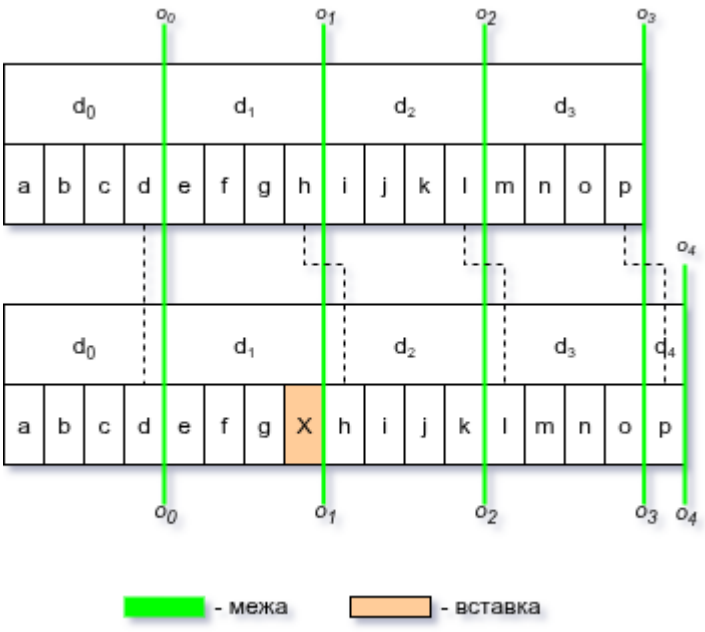


Рис. 5. Каскадне зміщення меж у FSC

1.2.2. Контенто-залежна фрагментація даних

Контенто-залежна фрагментація даних (з англ., Content-Defined Chunking; CDC) визначає межі фрагментів за властивостями вихідного потоку байтів. Читання потоку вихідних байтів здійснюється за допомогою ковзного вікна, а межа фрагменту встановлюється в кінці цього вікна коли його вміст задовольняє предикат *P*. На відміну від FSC, такий підхід до встановлення меж фрагментів усуває проблему каскадного зсуву меж, оскільки локальні вставки та вилучення впливають лише на межі у невеликій околиці змін, чим і досягається властивість стійкості меж. Методи CDC поділяють на дві великі групи: хешеві та безхешеві.

Хешеві методи CDC

У хешевих методах використовується ковзний хеш, що оновлюється з кожним наступним байтом у вікні сканування вихідного потоку байтів, як зображено на рис. 6. Одразу після оновлення хешу перевіряється предикат *P*, що, зазвичай, співставляє отриманий хеш з певною маскою *m*. У разі співставлення хеша з маскою, предикат *P* встановлює нову межу фрагменту.

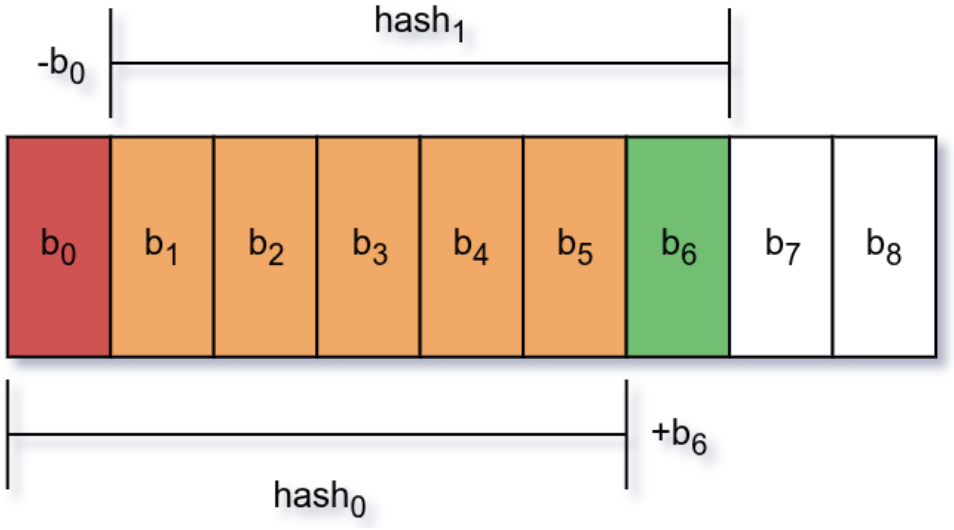


Рис. 6. Ковзний хеш

Не варто плутати ковзне хешування з хешуванням у конвеєрі дедублікації даних, адже це два різні процеси. Ковзне хешування не

призначене для ідентифікації вмісту фрагментів, а лише представляє модель байтів у вікні для співставлення з маскою.

Rabin CDC. Метод Rabin є класичним представником сімейства методів з ковзним вікном, з появою якого і було започатковано методи CDC [15]. Даний метод спирається на ідею поліноміального ковзного відбитка Рабіна з оновленням у ковзному вікні та перевіркою граничної умови за маскою. Саме Rabin CDC історично заклав базову модель CDC, однак нині цей метод вважається повільним відносно інших більш сучасних методів [16].

Нехай маємо ковзне вікно *w* довжиною *W*, поточний байт b_i та крайній байт b_{i-W+1}, тоді значення ковзного хешу h_i, що обчислюється за (23), де *x* – заздалегідь вирахуване значення внеску байта b_{i-W+1}, що покидає вікно. Як показано (24), межа фрагменту встановлюється коли g молодших бітів ковзного хешу h_i дорівнюють нулю. За умови наближеного до рівномірного розподілу значень ковзного хешу *h*, межа фрагменту може бути встановлена з ймовірністю $p = 2^{-g}$, а тому цільовий розмір фрагменту S_{avg} досягається шляхом вибору $g = \log_2(S_{avg})$.

$$h_i = (h_{i-1} - b_{i-W+1} \times x^{W-1}) \times x + b_i \; mod \; m = 0 \tag{23}$$

$$P_{Rabin} \coloneqq h_i \wedge (2^g - 1) = 0 \tag{24}$$

Для уникнення утворення занадто малих фрагментів, перевірка предикату P_{Rabin} не виконується допоки $i < S_{min}$. Відповідно з іншої сторони, якщо $i >= S_{max}$, то межа встановлюється примусово.

Сканування вихідного потоку байтів виконується побайтно, обчислення хешу *h(w)* і перевірка предиката P_{Rabin} у кожній позиції вікна дають лінійну алгоритмічну складність *O(L)*. Для кожного окремого байту реалізація Rabin CDC потребує одну операцію OR, дві операції XOR, дві операції бінарного зсуву та два звернення до таблиць за індексом.

До переваг методу Rabin відносять високий коефіцієнт дедублікації даних, однак, через низьку пропускну здатність даний метод майже не використовується у виробничих середовищах [17].

Gear CDC. На відміну від Rabin CDC, що використовує поліноміальний ковзний відбиток, підхід Gear CDC замінює сім операцій всього трьома: один бінарний зсув, одне звернення до таблиці за індексом та одне додавання. Це радикально зменшує вартість оновлення хешу у ковзному вікні та, як наслідок, підвищує пропускну здатність без зміни загальної моделі прийняття рішень про межу.

Нехай маємо заздалегідь підготовану таблицю випадкових значень G, тоді значення ковзного хешу h_i визначається за (25). Бінарний зсув затирає найстарший біт попереднього відбитка h_{i-1}, а додавання вносить інформацію про новий байт. У результаті отримуємо ковзний хеш з дуже малою кількістю операцій на крок.

$$h_i = (h_{i-1} \ll 1) + G[b_i] \tag{25}$$

Оскільки Gear CDC є прямим розвитком методу Rabin CDC, предикат P_{Gear} визначається ідентично до P_{Rabin}. Відповідно, обмеження довжини фрагментів S_{min} та S_{max} накладаються аналогічно до Rabin CDC.

Завдяки використанню простішого способу хешування ковзного вікна, метод Gear CDC дещо вирішив проблему низької пропускної здатності Rabin CDC [17], однак ціною гіршого коефіцієнту дедублікації через вищу варіативність довжин фрагментів, зокрема більшу кількість фрагментів розміру S_{max} [12].

Fast CDC. Даний метод є прямим продовженням Gear CDC з двома ключовими оптимізаціями. Перша оптимізація полягає у пропуску сканування байтів b_i, де $i < S_{min}$, а тому зменшуючи кількість операцій підрахунку ковзного відбитку та перевірок предикату P_{Fast}, при цьому ще підвищуючи пропуску здатність. Друга оптимізація вводить поняття нормалізації розміру фрагментів шляхом пом’якшення маски. Нехай маємо заданий рівень нормалізації N_{level}, згідно якого маска буде більш строгою до досягнення S_{avg} та відповідно менш строгою після досягнення S_{avg}. Таким чином, розрахунок ковзного хеша h_i лишається еквівалетним до Gear CDC, а предикат P_{Fast} визначається як:

$$P_{Fast} \coloneqq \begin{cases} h_i \wedge 2^{g+N_{level}} - 1 = 0, i \in [0, S_{avg}), \\ h_i \wedge 2^{g-N_{level}} - 1 = 0, i \in [S_{avg}, L). \end{cases} \quad (26)$$

Обмеження мінімальної довжини фрагментів S_{min} та примусове утворення фрагменту на S_{max} працює аналогічно Rabin та Gear CDC.

З появою методу Fast CDC, завдяки пропуску обробки ділянки до S_{min}, пропускна здатність фрагментації даних виросла у 2-2.5 рази [18]. Однак, навіть з введенням нормалізації розмірів фрагментів, проблему фрагментів довжиною S_{max} не було вирішено [18].

Безхешеві методи CDC

На відміну від сімейства хешевих методів CDC, де предикат встановлення межі P визначається збігом ковзного відбитка з маскою, безхешеві методи формулюють предикат P базуючись безпосередньо на локальних ознаках вмісту ковзного вікна. Типові приклади – детектування локальних екстремумів значень байтів або порівняння з опорними статистиками вікна (порядкові статистики, частоти символів або пар тощо). Інтуїтивно межа з'являється там, де накопичений контекст про вхідні байти істотно відрізняється від попереднього, тому навіть за локальних вставок/видалень межі фрагментів швидко ресинхронізуються, зберігаючи базову перевагу CDC над FSC.

Оскільки такі предикати обходяться без обчислення ковзного хешу, їхня вартість на обробку одного байту часто нижча. Водночас безхешеві методи потребують ретельного керування цільовим розміром фрагментів S_{conf} та шириною ковзного вікна W, оскільки надто вузьке вікно або контрастний предикат може призвести до надмірної кількості дрібних фрагментів, тоді як ділянки з низькою варіативністю байтів можуть спричинити велику кількість фрагментів розміром S_{max}. Тому на практиці використовують ті самі запобіжники, що й у хешевих CDC, а саме пропуск перевірок до S_{min}, встановлення примусових меж на S_{max}, а також легкі евристики для стабілізації розподілу довжин навколо S_{avg}.

AE (Asymmetric Extremum) CDC. Метод AE усуває дві ключові вади класичних методів з ковзним хешем, а саме низьку пропускну здатність через дорогі ковзні хеші та велику варіативність довжин чанків. Основна ідея AE полягає в тому, щоб шукати локальний екстремум у асиметричному вікні так, щоб встановлювати межу з мінімальною кількістю порівнянь. Це забезпечує високу швидкодію та низьку дисперсію довжин фрагментів задля збереження стійкості меж до локальних змін [19].

Нехай маємо вікно з фіксованим розміром W. У вихідному потоці байтів шукаємо байт b_{max} та його зміщення o_{max} такі, що $b_{max} = \max\{b_0, b_1, .., b_{i-1}, b_i\}$. Якщо останній такий байт було знайдено W ітерацій назад, то встановлюємо нову межу фрагменту зі зміщенням $o_{max} + W$. Предикат P_{AE} визначається як:

$$P_{AE} \coloneqq i - o_{max} - W = 0. \quad (27)$$

На відміну від розглянутих хешевих методів CDC, AE CDC використовує всього одну операцію порівняння на байт, завдяки чому досягається ще вища пропускна здатність. Тим не менш, зі збільшенням цільового розміру фрагменту S_{conf} падає коефіцієнт дедублікації D [19].

RAM (Rapid Asymmetric Maximum) CDC. Даний метод має дві фази виконання. Нехай маємо два вікна w_0 та w_1, тоді у перші фазі виконується пошук такого b_{max}, що $b_{max} = max(w_0)$, а у другій фазі межа фрагменту встановлюється при такому знайденому b_i, що $b_i >= b_{max}$. Предикат P_{RAM} визначається як:

$$P_{RAM} \coloneqq b_i \geq b_{max}. \quad (28)$$

Типово, реалізації методу RAM не використовують обмеження S_{min}, натомість його роль виконує довжина вікна w_0, в якому відбувається початковий пошук максимального значення. Обмеження S_{max} накладається як і в попередніх методах.

Даний метод, як і AE CDC, має всього одну операцію порівняння на байт, тому також має високу пропускну здатність відносно хешевих CDC.

Однак, даний метод чутливий до низькоентропійних даних, що може призвести до великої кількості фрагментів довжиною S_{max} [20].

SEQ (Sequential) CDC. Ще одним представником сімейства безхешевих методів CDC є SEQ CDC. Головною ідеєю цього методу є пошук у вихідній послідовності байтів впорядкованого вікна байтів w_{SEQ} довжиною L_{SEQ}, при знаходженні якого і встановлюється межа фрагменту [9]. Таким чином, предикат P_{SEQ} визначається як:

$$P_{SEQ} \coloneqq \exists j \in \left[S_{min} - L_{SEQ} + 1, L - L_{SEQ}\right] : \bigwedge_{i=j}^{L_{SEQ}} (b_{i-1} > b_i). \qquad (29)$$

Варто зазначити, що порядок шуканої послідовності може бути як за зростанням, так і за спаданням, однак вони є ексклюзивно виключними.

Крім вже зазначеного пошуку послідовності, даний метод має ще одну особливість, а саме пропуск послідовності байтів довжиною L_{skip} за умови, що протягом $N_{trigger}$ ітерацій не було знайдено жодної послідовності. Таким чином, даний метод пропускає нестабільні сегменти вихідного потоку байтів, при цьому підвищуючи пропускну здатність.

Як видно з (29), SEQ CDC також, використовує оптимізацію пропуску обробки перших S_{min} байтів, однак пошук послідовності починається на L_{SEQ} байтів раніше. Обмеження S_{max} накладається як і в попередніх методах.

SEQ CDC має дві операції порівняння на байт, а саме порівняння значень попереднього та поточного байтів, а також перевірка довжини накопиченої послідовності або перевірка довжини нестабільного сегменту байтів. Варто зауважити, що даний метод є чутливим до його налаштувань та може потребувати додаткових прогонок на цільових наборах даних з метою визначення ефективного порядку послідовностей [8].

1.3. Висновки до розділу 1

У даному розділі було досліджено процес дедублікації даних та його роль в умовах зростання об’ємів створення та копіювання даних у сучасному світі. Було вивчено основні етапи процесу дедублікації даних, першим і найвпливовішим з яких є етап фрагментації даних. Було

розглянуто два основні види фрагментації даних: фрагментація з фіксованим розміром фрагментів (FSC) та контенто-залежна фрагментація (CDC). Серед основних двох видів фрагментації виявлено принципову різницю їх роботи та доцільні варіанти використання.

Основний фокус проведеного аналізу був на контенто-залежній фрагментації даних, оскільки саме цей вид фрагментації доцільно використовувати з метою дедублікації даних. Було виділено та охарактеризовано дві групи CDC методів, а саме хешеві та безхешеві методи.

Також було проведено аналіз найбільш відомих методів у кожній із наведених груп методів CDC, окреслено принципи їхньої роботи, а також наведено їхні переваги і недоліки. Виявлені за результатами аналізу обмеження зумовлюють потребу в модифікації одного з цих методів з метою підвищення пропускної здатності фрагментації та мінімізації утворення аномально малих або великих фрагментів, що безпосередньо впливають на розміри індексів (рецептів) фрагментації даних і продуктивність реконструкції фрагментованих даних.

2. МОДИФІКАЦІЯ МЕТОДУ КОНТЕНТО-ЗАЛЕЖНОЇ ФРАГМЕНТАЦІЇ ДАНИХ

Запропонований у цьому розділі модифікований метод контенто-залежної фрагментації даних Twin CDC є прямим розвитком методу Fast CDC. Вибір базового методу саме з групи хешевих методів зумовлено двома факторами.

Першим фактором є баланс між простотою використання, прийнятною пропускною здатністю та коефіцієнтом дедублікації базового методу, наслідком чого є його поширене використання у комерційних продуктах, наприклад, Restic [21].

Другим фактором є використання базовим методом двох ключових оптимізацій, а саме пропуск обробки перших S_{min} байтів та введення механізму нормалізації розміру фрагментів. Попри доведену ефективність названих оптимізацій у порівнянні з методами попередніх поколінь [22], у запропонованому методі обидві ідеї зазнали істотних змін. Наведені зміни мають сприяти підвищенню пропускної здатності фрагментації та дедублікації даних, а також зведення до мінімуму кількості фрагментів, що мають довжини близькі до граничних значень S_{min} та S_{max}. Детальніше розглянемо базовий та модифікований методи.

2.1. Базовий метод Fast CDC

Як вже було розглянуто у розділі 1, у даному методі використовується хешування Gear в якості ковзного хешу. Пошук межі фрагменту відбувається у два етапи, починаючи з положення S_{min} у потоці байтів. Обидва етапи виконують ідентичні кроки, а саме для кожного байту b_i вираховується ковзний Gear хеш h_i згідно (25). Межа фрагменту встановлюється у випадку коли певна кількість молодших бітів k дорівнюють нулю.

Різниця між наведеними етапами полягає саме у кількості молодших бітів k. Перший етап сканує потік у проміжку $[S_{min}; S_{avg})$, і при цьому враховує більшу кількість молодших бітів k_{strict}, другий же етап виконує сканування у проміжку $[S_{avg}; L)$, при цьому враховуючи вже меншу кількість молодших бітів k_{lax}.

Таким чином досягається нормалізація розмірів фрагментів, адже метод виконує пошук більш строго (використовуючи *strict mask*) до досягнення S_{avg} та менш строго (використовуючи *lax mask*) після досягнення S_{avg}. На рис. 7 та на лістингу 1 (додаток 2) зображено повний метод Fast CDC, де з 8 по 13 стрічку виконується перший етап та з 14 по 18 стрічку виконується другий етап. В разі, якщо на жодному з етапів не вдалося знайти такий хеш, що задовольняє більш строгу або менш строгу умови, межа встановлюється примусово на S_{max}.

Ймовірність знаходження такого ковзного хешу h_i, що, в залежності від поточного етапу, задовольняє одну з масок обернено пропорційна кількості бітів k, що мають бути рівними нулю відповідно до:

$$p_{strict} = 2^{-k_{strict}}, \tag{30}$$

$$p_{lax} = 2^{-k_{lax}}. \tag{31}$$

Підбір значень k_{strict} та k_{lax} цілком залежить від заданого цільового розміру фрагментів S_{conf}, а саме від середнього очікуваного розміру фрагментів S_{avg}. З розрахунку ймовірності події p (30), (31) випливає, що математичне сподівання розміру фрагментів $S_{avg} = 2^k$. Отже, базове значення k розраховується за (32), а відповідні k_{strict} та k_{lax} за:

$$\mathrm{k} = \lceil \log_2(S_{avg}) \rceil, \tag{32}$$

$$k_{strict} = k + 1, \tag{33}$$

$$k_{lax} = k - 1. \tag{34}$$

Очевидно, що ймовірності в обох етапах нерівні між собою, адже $k_{strict} > k_{lax}$, а отже кількість фрагментів розміром більшим за S_{avg} буде переважати кількість фрагментів з розміром меншим за S_{avg}. Це свідчить про

характер розподілу розмірів фрагментів, який відповідно буде зміщений у більшу сторону від S_{avg}.

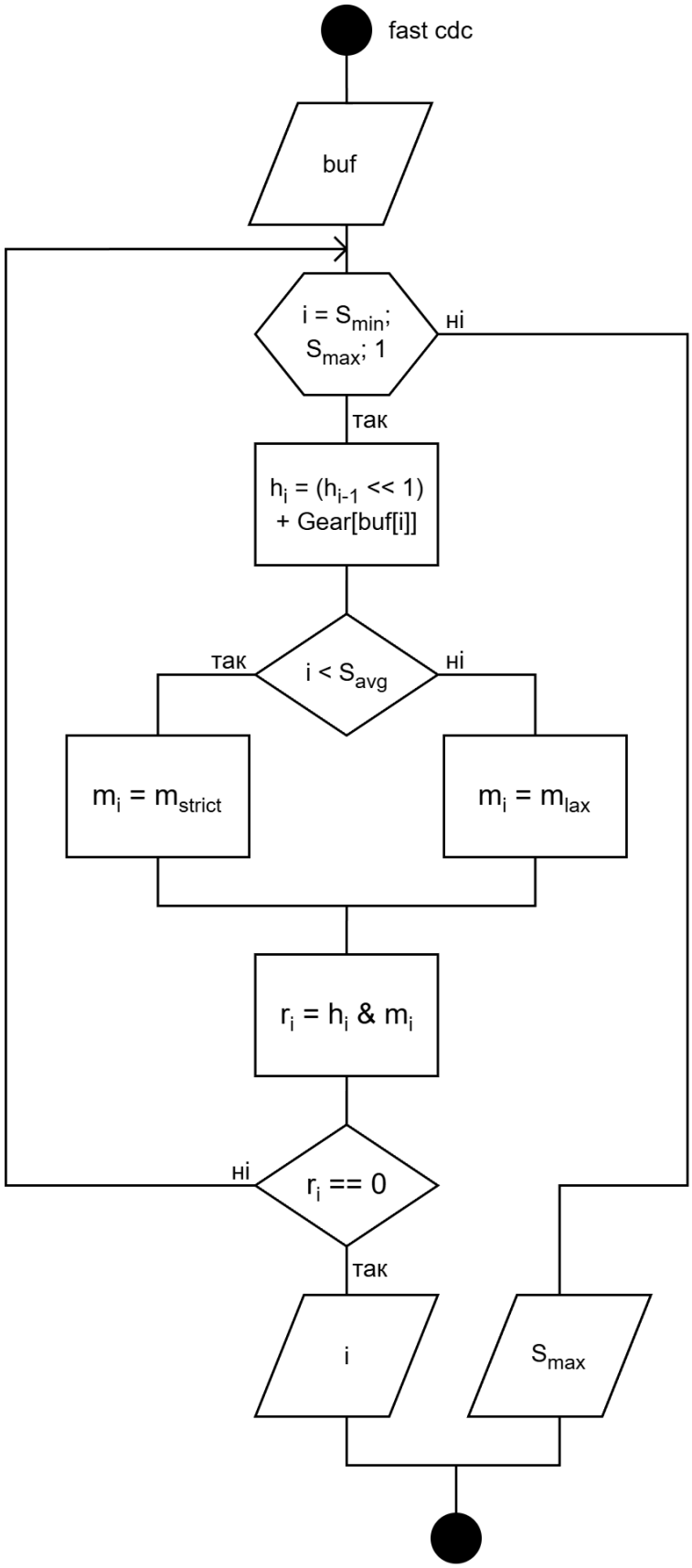


Рис. 7. Блок-схема методу Fast CDC

2.2. Модифікований метод Twin CDC

Метод контенто-залежної фрагментації даних Twin CDC є модифікацією методу Fast CDC. Ключовими відмінностями даної модифікації є використання двонаправленого сканування вихідного потоку байтів, використання двох Gear-таблиць та встановлення альтернативних меж фрагментів. На рис. 8 зображено блок-схему даного методу.

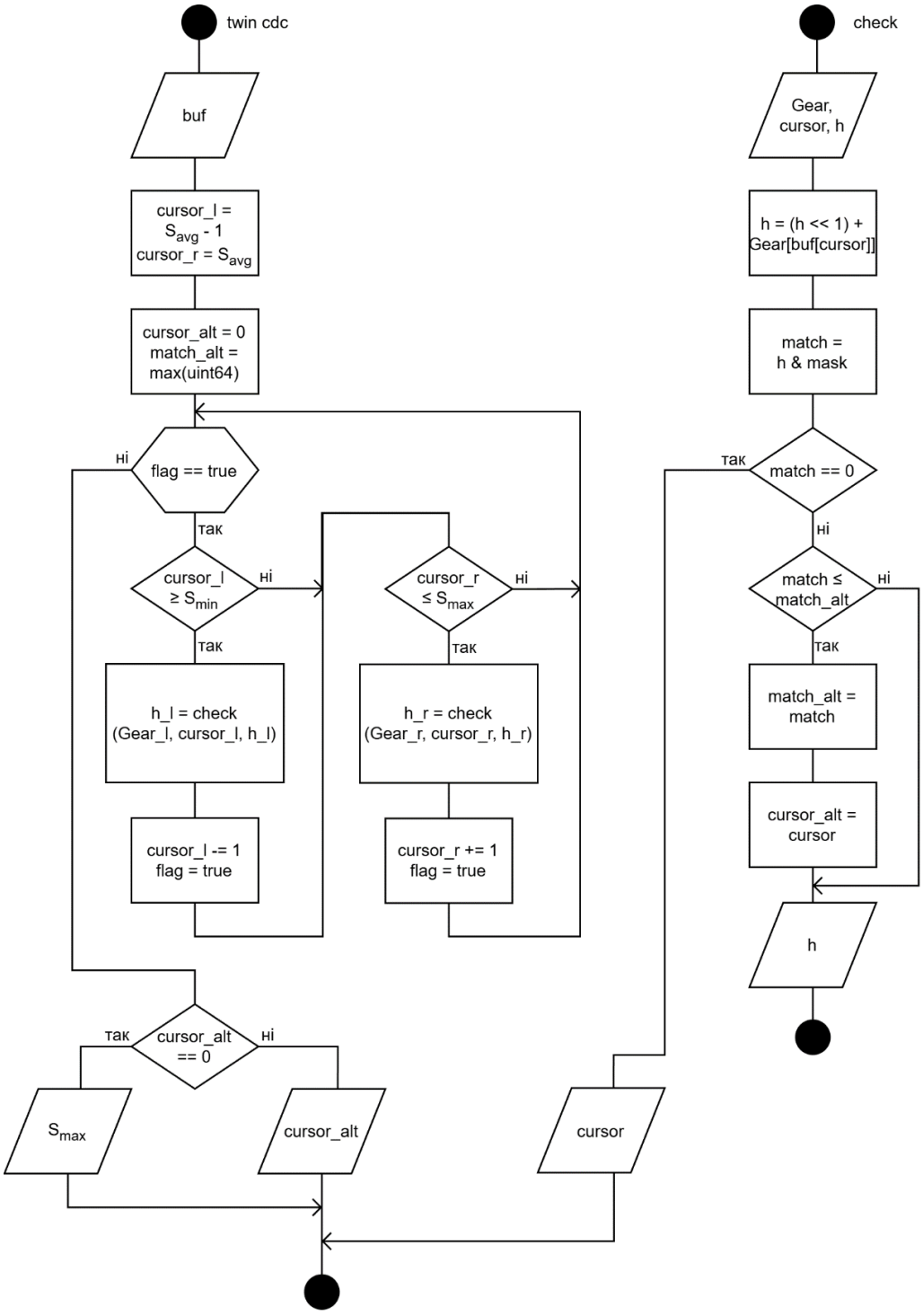


Рис. 8. Блок-схема методу Twin CDC

Як і базовий метод, Twin CDC конфігурується цільовим розміром фрагментів S_{conf} та заданим рівнем нормалізації. Крім стандартних параметрів, Twin CDC також має налаштування режиму роботи, а саме використання однієї або двох незалежних таблиць Gear для обчислення ковзного хешу. Як видно з рис. 8, а також з лістингу 2 (додаток 2), предикат встановлення межі P_{Twin} якого є подібним до предикату P_{Fast} (26) та визначається як:

$$P_{Twin} \coloneqq \begin{cases} \exists i \in [0, L): h_i \wedge (2^{g-N_{level}} - 1) = 0, \\ \exists i, j \in [0, L): h_i = min(h_j), \\ S_{max}. \end{cases} \quad (35)$$

2.2.1. Двонаправлений пошук меж фрагментів

Ключова ідея двонаправленого пошуку меж фрагментів полягає у радіальному перегляді кандидатних позицій на встановлення межі, починаючи від опорної точки. Як показано на рис. 9 та 10, на відміну від Fast CDC та інших розглянутих методів, пошук в яких відбувається лише вперед з початку (зі зсуву S_{min}) до S_{max}, у модифікованому методі Twin CDC пошук починається з певного зсуву всередині поточного буферу вихідних байтів і одночасно розширює зону перевірки у двох протилежних напрямках: ліворуч у сторону S_{min} і праворуч у сторону S_{max}. Такий підхід до пошуку має забезпечити пріоритетне опрацювання позицій, найбільш ймовірно пов’язаних із очікуваним середнім розміром фрагменту.

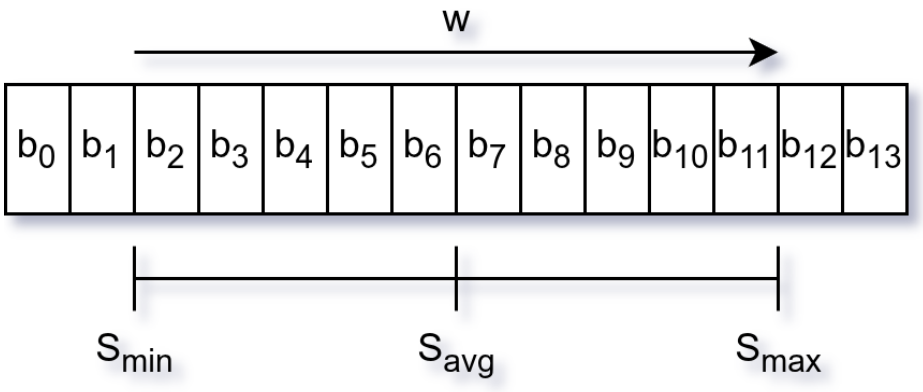


Рис. 9. Вікно та напрямок сканування потоку байтів Fast CDC

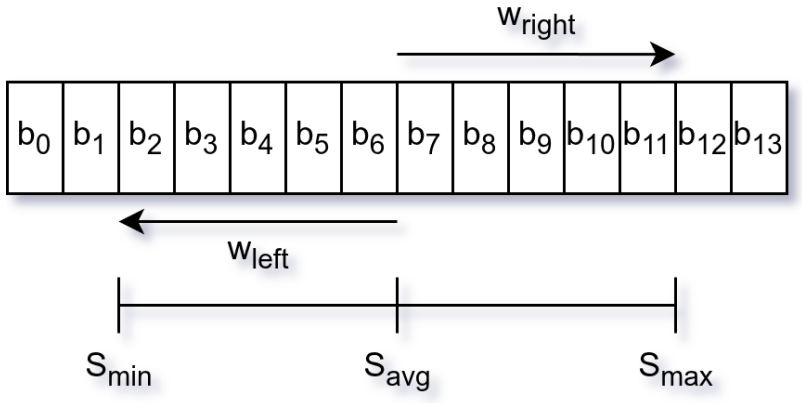


Рис. 10. Вікно та напрямок сканування потоку байтів Twin CDC

Опорною точкою є зсув o_{mid}, що за значенням рівний середній очікуваній цільовій довжині фрагменту S_{avg}. Відштовхуючись від зсуву o_{mid} задаються два вікна пошуку межі: ліве w_{left} та праве w_{right} відносно зсуву o_{mid}, при чому кожне з вікон належить відповідним проміжкам, лівому та правому відповідно:

$$\forall i \in [S_{min}, o_{mid}): w_i \in w_{left}, \tag{36}$$

$$\forall j \in [o_{mid}, S_{max}): w_j \in w_{right}. \tag{37}$$

Пошук у кожному з вікон відбувається за допомогою курсорів c_{left} та c_{right}, за допомогою яких зчитуються значення поточних байтів на кожній ітерації, при чому c_{left} рухається в межах $o_{mid} - 1 \rightarrow S_{min}$, а c_{right} рухається в межах $o_{mid} \rightarrow S_{max}$. Як видно з лістингу 2 (додаток 2), за одну ітерацію модифікований метод виконує сканування одразу двох байтів, по одному байту з лівого та правого вікон. Такий підхід може нагадати класичну оптимізацію розгортання циклів, згідно якої кількість ітерацій зменшується за рахунок збільшення кількості операцій на одну ітерацію [23]. Однак, на відміну від розгортання циклів, де напрямок ітерування залишається одностороннім та послідовним, у Twin CDC напрямки лівого та правого вікон протилежні. В той же час, твердження про виконання двох операцій за одну ітерацію є істинним в контексті запропонованої модифікації методу.

Запропонований метод припиняє пошук та встановлює межу за умови спрацювання предикату P_{Twin} у будь-якому з вікон. В разі спрацювання P_{Twin}

в обох вікнах одночасно, перевага надається лівому вікну, як показано на стрічці 16 лістингу 2 (додаток 2).

Варто також зазначити, що у запропонованій модифікації також дотримано обмеження мінімального та максимального розмірів фрагментів, оскільки задані курсори ніколи не виходять за межі проміжку [S_{min}, S_{max}). У випадку коли предикат P_{Twin} не було виконано у жодному з вікон на всьому проміжку [S_{min}, S_{max}), встановлюється альтернативна межа фрагменту.

2.2.2. Встановлення альтернативних меж фрагментів

У більшості методів, що були розглянуті у розділі 1, в разі якщо не спрацьовує предикат встановлення межі фрагменту P до досягнення максимального допустимого розміру фрагментів S_{max}, тоді межа фрагменту встановлюється на зсуві S_{max}. У той же час, базовий метод Fast CDC, намагається уникнути досягнення S_{max} без встановлення межі шляхом використання маски з меншою кількістю бітів k_{lax}, завдяки чому і збільшується ймовірність встановлення межі.

У модифікованому методі використовується єдина маска k, що за значенням є еквівалентною масці k_{lax} у Fast CDC. Натомість, з метою уникнення аномально малих та великих фрагментів у Twin CDC використовується встановлення альтернативних меж. Нехай маємо значення m_i, що є еквівалентним результату накладення маски k на хеш h на кожному зсуві у проміжку $[S_{min}, S_{max})$. *Тоді,* замість двох фіксованих масок, метод, крім пошуку такого значення ковзного хешу h, що задовольняє предикат P_{Twin}, також виконує пошук мінімального значення m_i. Таким чином, за умови невиконання предикату P_{Twin} у проміжку $[S_{min}, S_{max})$, межа встановлюється на такому зсуві o_i, що $m_i = min(m)$. Фактично, це означає, що вторинна умова встановлення межі не визначена наперед, а встановлюється на зсуві, де було досягнуто мінімальне значення результату накладення маски m. Саме таким чином забезпечується оптимальне наближення до цільового розміру фрагментів, адже замість встановлення

межі фрагменту на зсуві S_{max} метод надасть перевагу встановленню межі в точці локального мінімуму функції ковзного хешу, яка, очевидно, ніколи не досягає S_{max}.

Оскільки ковзний хеш Gear є псевдовипадковим для довільного потоку даних, такий локальний мінімум із високою ймовірністю відповідає унікальній комбінації байтів, яка практично задовольняє умови маски. Можна припустити, що така комбінація є характерним шаблоном даних, хоча і не повністю відповідає предикату P_{Twin}, тобто не надає повний збіг маски, але вирізняється на фоні сусідніх. Таким чином, встановивши межу фрагменту саме там, метод, ймовірно, утворює фрагмент у місці, близькому до точки контрастної зміни даних. Це може сприяти кращому вилученню дубльованих фрагментів, оскільки фрагменти відрізані по найбільш значущих місцях.

Встановлення альтернативної межі за локальним мінімумом функції ковзного хешу також має стабілізувати характеристику розмірів фрагментів. Якщо порівняти модифікований метод з, наприклад, Gear CDC, що працює без нормалізації, Twin CDC завдяки встановленню альтернативних меж має звужувати фактичний розподіл розмірів фрагментів. Всі фрагменти, що були утворені з альтернативної межі гарантовано будуть дещо меншими за S_{max}, оскільки майже завжди знайдеться якийсь субоптимальний кандидат до того, як досягнуто S_{max}. Це і було підтверджено методом Fast CDC, де вперше введено механізм нормалізації розмірів фрагментів шляхом штучного послаблення умови після досягнення середньої довжини, аби змусити межу виникнути до досягнення S_{max} [18]. Однак, у Fast CDC нормалізація реалізована через зменшення кількості ненульових біт на другому етапі пошуку межі фрагменту, тоді як Twin CDC досягає схожого ефекту без додаткових порогів, де будь-яке мінімальне значення хешу спричинить встановлення межі фрагменту.

2.2.3. Використання окремих Gear таблиць

У вже розглянутих CDC методах з ковзним хешем використовується єдина ковзна хеш функція для всього потоку даних, наприклад, Rabin або Gear [12]. У модифікованому методі Twin CDC також використовується ковзний хеш Gear, однак для хешування вихідних байтів лівого та правого вікон можуть бути використані дві окремі таблиці Gear, що містять різні випадкові значення для кожного можливого значення байту.

Нехай маємо ймовірність події встановлення межі p, що визначається за (30), тоді обернена ймовірність визначається як $p' = (1 - p)$. Отже, ймовірність не встановити межу на відстані d починаючи з опорної точки S_{avg} визначатиметься як показано (38), оскільки на кожній ітерації модифікований метод виконує по одній перевірці ковзного хешу у правому та лівому вікнах. Важливо зауважити, що наведена оцінка припускає незалежність зазначених подій.

$$p'_d = (1-p)^d \times (1-p)^d = (1-p)^{2d} \tag{38}$$

Припустимо, що знайдеться такий окіл вихідного потоку байтів, що у кожній точці цього околу значення ковзного хешу у лівому вікні h_{left} та у правому вікні h_{right} будуть корелювати між собою. Така кореляція може бути позитивною, тобто давати однаково наближені до нуля значення накладення маски m, так і однаково віддалені від нуля значення, тобто бути негативною. Позитивна кореляція ніяк не впливає на швидкість встановлення межі чи її віддалення від опорної точки S_{avg}, оскільки умова спрацювання предикату P_{Twin} хоча б в одному з вікон є достатньою для встановлення межі. З іншої сторони, негативна кореляція фактично підвищуватиме ймовірність p'_d, оскільки послідовність однаково далеких від нуля результатів накладення маски в обох вікнах m_{left} та m_{right} означає більший локальний мінімум, що буде використаний для встановлення альтернативної межі.

Виникнення кореляції можливо, наприклад, коли у наведеному околі вихідного потоку байтів результати накладення маски на ковзні хеші у

вікнах симетричні та складають довгі ділянки без необхідних нульових бітів у значеннях хешів, а отже і дають відносно великі локальні мінімуми.

Таким чином, у разі негативної кореляції значень ковзного хешу у лівому і правому вікнах ймовірність невстановлення межі визначатиметься однією подією, а отже таку ймовірність можна розрахувати за:

$$p'_d = (1-p)^d. \tag{39}$$

Використання двох окремих Gear таблиць для розрахунку ковзного хешу у лівому та правому вікнах призначене саме для декореляції значень ковзного хешу у цих вікнах. Якщо у одному з вікон спостерігається серія невдалих накладень маски на ковзний хеш, відповідна операція в іншому вікні є повністю незалежною, а отже може результувати кращим значенням m.

Таким чином, декореляція значень ковзного хешу має зменшити ризик виникнення синхронних невдалих накладень масок на ковзний хеш, наприклад, у структурованих або монотонних даних, де використання єдиної Gear таблиці може результувати спільною серією невдалих значень ковзного хеша в обох вікнах.

2.2.4. Збереження стійкості меж

Вже розглянуті односторонні Gear-подібні методи мають деяке упередження щодо встановлення меж до досягнення S_{avg}, оскільки байти, що знаходяться на початку вихідного потоку даних (або в правому околі S_{min}), мають найбільший вплив на подальший пошук межі, а тому потенційні межі в околі S_{avg} мають ймовірність так і не бути розглянутими. Будь-яка локальна правка спочатку впливає саме на ранній контекст, і однобічний метод відгукується на неї вже на початку інтервалу, потенційно зміщуючи межу та всі наступні орієнтири [24].

У Twin CDC ранні позиції не мають пріоритету, оскільки першість надається позиціям у безпосередній околиці S_{avg}. Саме тому, навіть якщо локальна правка вплинула на структуру байтів поблизу початку інтервалу,

перша межа з високою ймовірністю буде знайдена ближче до середини. Це безпосередньо і сприяє стійкості меж, адже введена локальною зміною різниця у межах передається не ланцюжком, а затухає в околі одного фрагмента.

2.3. Висновки до розділу 2

У цьому розділі було сформульовано та обґрунтовано модифікацію методу контенто-залежної фрагментації Twin CDC, що є прямим розвитком методу Fast CDC. Дана модифікація орієнтована на звуження розподілу довжин фрагментів без введення додаткових етапів або порогів перевірки, при цьому підвищуючи пропускну здатність та зберігаючи коефіцієнт дедублікації даних.

У запропонованому методі було поєднано три ключові модифікації, що гармонійно доповнюють одна одну. Завдяки двонаправленому пошуку межі, методом пріоритизуються околиці цільового середнього розміру фрагментів S_{avg}, що має результувати більшою кількістю фрагментів, розмір яких рівномірно розподілений в околі S_{avg}. Встановлення альтернативних меж фрагментів має на меті зменшити кількість фрагментів, що були утворені примусово за обмеженням розміру фрагментів S_{max}. Використання окремих Gear таблиць покликане зменшити негативні кореляції під час сканування вихідного потоку байтів, таким чином знижуючи ймовірність довгих синхронних ділянок небажаних залишків результатів накладання маски у структурованих або монотонних даних, а отже, підтримуючи модифікацію встановлення альтернативних меж фрагментів.

Підсумовуючи, Twin CDC теоретично має підвищити пропускну здатність фрагментації даних завдяки обробці двох байтів на ітерацію та звести до мінімуму крайні випадки із примусовим встановленням межі фрагментів на S_{max} завдяки встановленню альтернативних меж, а також меншу чутливість до структурованих шаблонів у вихідному потоці даних завдяки декореляції лівого та правого вікон.

3. ОСОБЛИВОСТІ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МОДИФІКОВАНОГО МЕТОДУ КОНТЕНТО-ЗАЛЕЖНОЇ ФРАГМЕНТАЦІЇ ДАНИХ

3.1. Функціональні вимоги

Імплементація розглянутих методів CDC. З метою подальшого аналізу розглянутих методів фрагментації даних у порівнянні з модифікованим методом, розроблювана програмна реалізація має містити імплементації всіх розглянутих у даній роботі методів фрагментації даних, включаючи:

- Fixed-Size;
- Rabin;
- Gear;
- Fast;
- Asymmetric Extremum;
- Rapid Asymmetric Maximum;
- Sequential.

Задля повної взаємозамінюваності будь-якого методів іншим у будь-яких сценаріях аналізу та порівняння реалізованих методів, всі імплементації кожного з наведених методів мають бути реалізовані з використанням спільного програмного інтерфейсу (API) для фрагментації даних.

Імплементація модифікованого методу Twin CDC. Розроблювана програмна реалізація має містити повну імплементацію модифікованого методу контенто-залежної фрагментації даних Twin CDC, що відповідає опису запропонованого методу у розділі 2. Реалізація модифікованого методу може містити особливості, що характерні обраній мові програмування, однак головні риси модифікованого методу, такі як двонаправлений пошук меж фрагментів, встановлення альтернативних меж фрагментів, а також можливість використання двох окремих таблиць Gear

мають бути збережені та імплементовані відповідно до лістингу 2 (додаток 2). Крім того, реалізація модифікованого методу також має використовувати спільний програмний інтерфейс для фрагментації даних.

Потокова обробка даних. Розроблювані програмні реалізації методів мають підтримувати контенто-залежну фрагментацію даних великого об’єму без завантаження всього вмісту вхідних даних у пам’ять, в тому числі даних об’ємом більше 2 гігабайтів, що складає типове обмеження розміру одного масиву даних для завантаження у пам’ять [25]. Тому, з метою уникнення завантаження вхідних даних у пам’ять цілком, програмна реалізація має використовувати абстракції потокової обробки даних обраної мови програмування.

Конфігурація фрагментації даних. Програмна реалізація має підтримувати конфігурування будь-якого обраного методу фрагментації даних, не обмежуючись спільними налаштуваннями цих методів та надаючи повний контроль користувацькому коду над усіма конфігураційними значеннями, що характерні кожному окремому методу.

Опціональна емісія фрагментів. З метою отримання точних метрик продуктивності методів фрагментації даних, їхні реалізації не мають містити логіки утворення фрагментів. Натомість, результатом роботи кожного з методів, відповідно до їх спільного програмного інтерфейсу, має стати зсув, на якому встановлюється межа фрагменту для заданого буферу даних. Емісія фрагментів даних має бути реалізована окремо та використана лише у випадках, коли необхідно використати вміст фрагментів для отримання певної метрики.

3.2. Нефункціональні вимоги

Детермінізм та ідемпотентність. За умови фіксованої конфігурації кожного обраного методу для тих самих вхідних даних результат фрагментації таких даних має бути ідентичним при кожному виконанні

фрагментації, незалежно від архітектури центрального процесора, порядку зберігання байтів у пам’яті (big або little endianess) та операційної системи, в межах якої виконується фрагментація даних, включаючи Linux, macOS та Windows.

Безпека виконання. Процес фрагментації даних має бути неперервним та не спричиняти неочікуваних переривань, виключних ситуацій або несподіваних завершень роботи. Всі необхідні перевірки, що можуть призвести до наведених ситуацій мають відбуватися до безпосереднього початку фрагментації даних, тобто на етапі конфігурування будь-якого обраного методу.

3.3. Обраний набір технологій

3.3.1. Мова програмування

Для реалізації заданих вимог було обрано мову програмування C# 13.0 на основі платформи .NET 9. Сучасний JIT-компілятор RyuJIT, у якому застосовується багаторівнева компіляція та профіле-керовані оптимізації часу виконання у поєднанні з розширеною девіртуалізацією викликів методів [26], що має забезпечити необхідний баланс між безпекою пам’яті й продуктивністю для інтенсивних операцій над потоками байтів, характерних для CDC методів. Крім того, стандартна бібліотека платформи .NET має готові абстракції, що необхідні для виконання I/O операцій.

3.3.2. Механізм отримання метрик продуктивності

Для отримання метрик продуктивності кожного з перелічених методів фрагментації даних було обрано пакет BenchmarkDotNet, що фактично є стандартним механізмом для написання та виконання бенчмарків у екосистемі платформи .NET [27]. Обраний пакет забезпечує статистично коректні вимірювання, ізоляцію середовища виконання бенчмарків, розділення фаз прогріву середовища та зняття метрик, а також широкий

набір конфігурацій і засобів діагностики, необхідних для відтворюваних експериментів у роботі [27].

Імплементація бенчмарків. Кожен бенчмарк представляється набором методів класу, що містить всі необхідні інструкції для виконання роботи, метрики продуктивності якої і заміряються. Кожен з оголошених методів має мати атрибут *Benchmark*, завдяки якому середовище виконання бенчмарків знаходить методи для зняття метрик. Один з бенчмарків може бути помічений як еталонний, таким чином результати решти бенчмарків будуть порівняні з еталонним автоматично. Приклад оголошення двох бенчмарків в одному класі зображено у лістингу 3 (додаток 2), при чому бенчмарк *DoWork_First* є еталонним.

Вимірювання метрик продуктивності. Метрики продуктивності вимірюються шляхом багатократного виконання методу бенчмарку, тобто за результатами певної кількості повторень вираховується середній час виконання методу бенчмарку, похибка вимірювання, стандартне відхилення тощо. BenchmarkDotNet автоматично визначає необхідну кількість повторень на етапі прогріву оточення бенчмарку [27]. Через специфіку роботи JIT компілятора, декілька перших ітерацій результуватимуть гіршими метриками, оскільки саме під час першого виконання виконується JIT компіляція з IL у машинний код, при чому первинна компіляція може не дати достатнього рівня низькорівневої оптимізації. Цей випадок називають tier-0 compilation або компіляція нульового рівня, на якому первинною задачею JIT є мінімізація затримки виклику методу вперше [28]. Однак, за умови що метод бенчмарку буде класифікований як найбільш вимоглива ділянка коду (так званий hot code execution path або hot path), тоді JIT компілятор перекомпілює такий метод із застосуванням всіх доступних оптимізацій (tier-1). Tier-1 компіляція займає більше часу, ніж tier-0, однак додані накладні витрати на оптимізацію hot path методу виправдовуються меншим часом виконання заданої вимогливої ділянки коду. Отже, щоб отримати найбільш оптимізовану версію методу бенчмарку,

BenchmarkDotNet спершу виконує прогрів бенчмарку, повторно здійснюючи виклики методу бенчмарку допоки не буде застосована tier-1 компіляція [28]. Очевидно, що продуктивність бенчмарку на етапі прогріву не враховується у результуючі значення метрик продуктивності, однак під час фактичного виконання бенчмарку вже використовується найбільш оптимізована версія методу бенчмарку.

Параметризація бенчмарків. Для системного порівняння різних методів або методів з різними конфігураціями використовується параметризація бенчмарків. Даний аспект дозволяє один раз описати бенчмарк та визначити його змінні властивості, наприклад, метод фрагментації, цільовий розмір фрагментів S_{conf}, або файли різних розмірів, що підлягають фрагментації. Під час виконання оголошених бенчмарків пакет автоматично генерує всі можливі комбінації заданих параметрів та виконує кожен бенчмарк з кожною такою комбінацією параметрів, завдяки чому і досягається параметризація. Для застосування параметризації пропонується два способи. Одним із способів є використання атрибуту *Params*, що дозволяє оголосити всі значення параметру разом з оголошенням самого параметру. Однак, через обмеження мови програмування C#, такі атрибути можуть містити лише константи, тобто числа, фіксовані набори, стрічки або їх масиви. В разі необхідності використання змінних, комплексної послідовності створення параметрів або їх великої кількості пропонується використання альтернативного атрибуту *ParamsSource*, що дозволяє делегувати деталі створення параметрів окремому методу. Лістингом 4 (додаток 2) зображено використання обох способів параметризації бенчмарків, де значення *InlineParam* задається разом з оголошенням параметру, а значення *ExternalParam* задаються окремим методом *EnumerateValues*. Варто зауважити, що BenchmarkDotNet не підтримує визначення еталонних параметрів, тому за необхідності порівняння результатів бенчмарку з різними параметрами можна застосувати механізм користувацьких колонок.

Підготовка оточення. У випадку, коли для виконання бенчмарків необхідно проініціалізувати або звільнити деякі ресурси до початку або по закінченню виконання цих бенчмарків, BenchmarkDotNet дозволяє оголосити чотири методи, що є ініціалізаторами та деструкторами середовища виконання бенчмарків. Кожен з цих методів дозволяє виконати певні підготовчі або завершальні інструкції та має бути позначений відповідним атрибутом в залежності від необхідного випадку використання:

- *GlobalSetup (ініціалізатор бенчмарку)* для ініціалізації ресурсів до виконання всіх бенчмарків;
- *IterationSetup* (ініціалізатор ітерації) для ініціалізації ресурсів до виконання кожної з ітерацій кожного бенчмарку;
- *GlobalCleanup (очисник бенчмарку)* для звільнення ресурсів після виконання всіх бенчмарків;
- *IterationCleanup* (очисник ітерації) для звільнення ресурсів після виконання кожної ітерації кожного бенчмарку.

Таким чином, всі підготовчі інструкції можуть бути винесені в окремі методи, що будуть викликані середовищем виконання бенчмарків відповідно до поточного етапу виконання. Варто також зазначити, що час та пам’ять які були використані під час виконання будь-якого з наведених методів не враховуються у результатах бенчмарків продуктивності, а отже наведений механізм підготовки оточення додатково сприяє точності отриманих метрик. На рис. 11 зображено послідовність ініціалізації, виконання та завершення бенчмарків. На практиці ці методи застосовуються для завантаження тестових наборів даних, підготовки буферів та попереднього обчислення допоміжних структур, наприклад, хеш-таблиць або індексів, щоб уникнути спотворення результатів. Окрему увагу доцільно приділяти детермінізму: ініціалізація має виконуватися однаково між запусками, а звільнення ресурсів — гарантувати відсутність побічних ефектів для наступних ітерацій. Такий підхід забезпечує відтворюваність експериментів і полегшує інтерпретацію метрик.

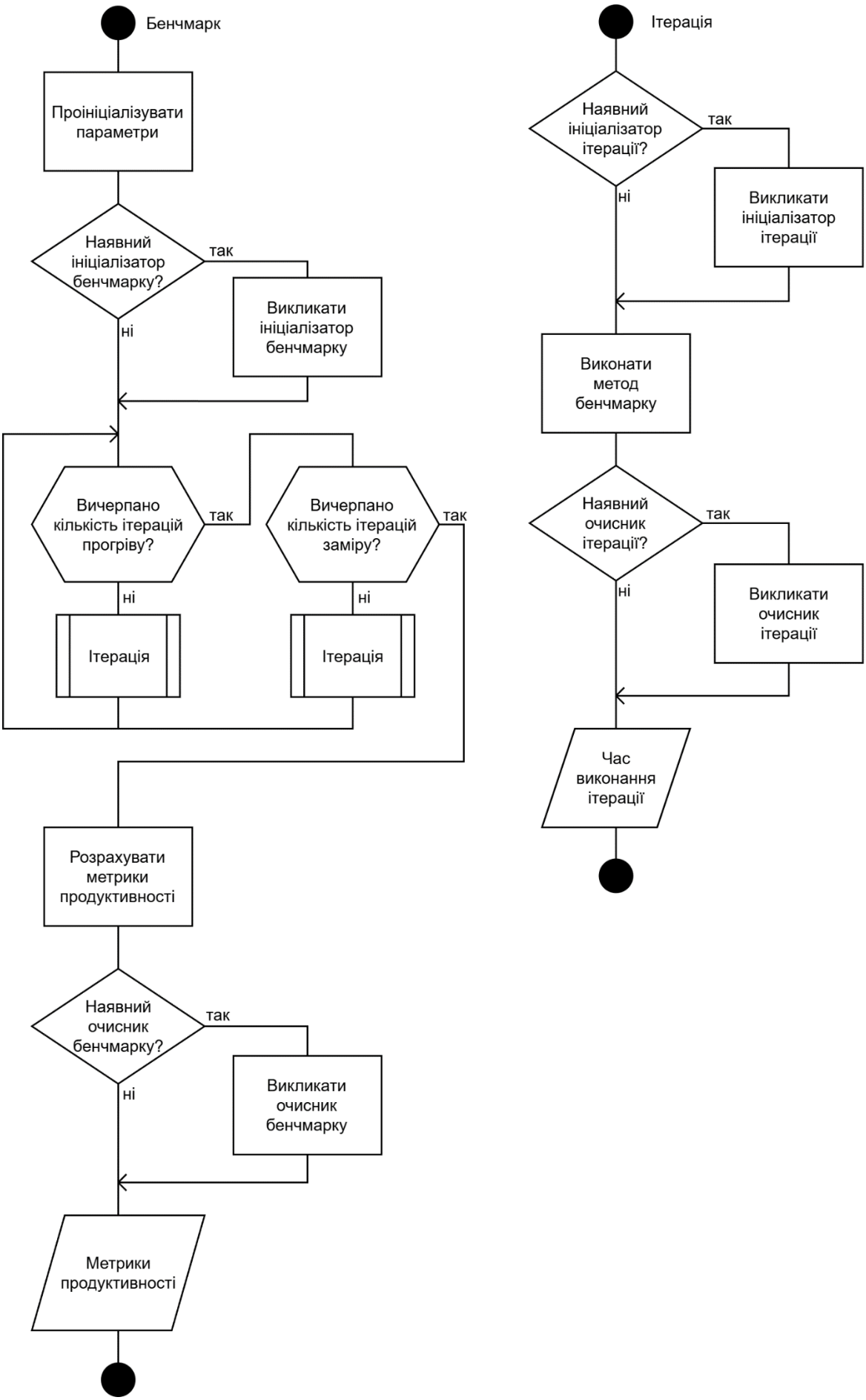


Рис. 11. Блок-схема послідовності виконання бенчмарків

Інструменти діагностики. З метою отримання діагностичних даних та метрик, які не вимірюються за замовчуванням, застосовують інструменти діагностики. Наразі, BenchmarkDotNet постачає декілька таких інструментів, однак більшість з них мають обмежену підтримку серед операційних систем, в контексті яких виконуються бенчмарки [27]. Одним із базових інструментів діагностики є діагностика аллокацій пам'яті, що реалізовано класом *MemoryDiagnoser*. Під час виконання бенчмарків, даний інструмент аналізує загальну кількість виділеної пам'яті та кількість відпрацювань GC. Ці метрики дозволяють порівняти споживання пам'яті різними методами, визначити частоту відпрацювань GC у кожному з поколінь та, відповідно, кількість пауз у виконанні інструкцій кожного з бенчмарків.

Користувацькі колонки. За допомогою користувацьких колонок можна автоматизувати підрахунок деяких користувацьких метрик, що базуються на вбудованих метриках. Таким чином, крім вбудованих метрик продуктивності (середній час виконання, стандартне відхилення, похибка), є можливість реалізувати власні підрахунки та додати їх до вбудованих метрик. Крім вбудованих метрик, для створення та підрахунку користувацьких метрик також доступні параметри виконання кожного бенчмарку, а отже, доступно більше контексту про бенчмарк, для якого вираховується користувацька метрика. Для створення користувацької колонки достатньо створити власний клас та реалізувати ним інтерфейс *IColumn*, після чого додати екземпляр цього класу до списку колонок у конфігурації бенчмарків.

3.3.3. Механізм отримання метрик дедублікації даних

На відміну від процесу отримання метрик продуктивності, в якому швидкість виконання фрагментації оцінюється завдяки багатократному повторенню фрагментації на обраній парі методу та вхідного файлу, для отримання метрик дедублікації достатньо виконати фрагментацію лише

один раз для аналогічної пари. Тим не менш, для правильного підрахунку метрик дедублікації даних необхідно виконати серію підрахунків.

З метою простого компонування підрахунків було застосовано пакет AnyKit.Pipelines. Даний пакет дозволяє складати конвеєри обробки даних з окремих елементів, при чому кожен елемент має контроль над викликом наступного елементу. Таким чином досягається універсальність отримання даних, тобто будь-який етап обробки даних може використовувати дані, отримані у попередній або у наступній фазі.

Обраний пакет компонування дозволяє інкапсулювати послідовність з декількох фаз обробки даних в єдиний екземпляр делегату, при цьому не розкриваючи деталей про окремі застосовані фази. Так, після компіляції конвеєра обробки даних, для зовнішнього споживача цього конвеєра доступна лише публічна сигнатура цього конвеєра, тобто тип аргументу та тип результату. Кожна фаза конвеєра обробки даних оперує вже згаданим контекстом, а також делегатом виклику наступної фази.

3.4. Архітектура розробленого програмного забезпечення

Розроблений програмний комплекс *ChunkIt* є спеціалізованим засобом для зняття метрик та подальшого візуального аналізу якості контенто-залежної фрагментації та дедублікації даних. Архітектура побудована за модульним принципом і складається з набору проєктів, кожен з яких містить деталі імплементації одного з аспектів процесу дедублікації даних, зняття та візуалізації метрик, або спільної інфраструктури. Такий підхід спрощує повторне використання компонентів, розширення набору методів фрагментації та метрик, а також забезпечує прозорість експериментального конвеєра – від конфігурації запуску до візуалізації отриманих метрик для їх подальшого аналізу. На рис. 12 зображено діаграму модулів розробленого програмного комплексу та їх залежності одне від одного.

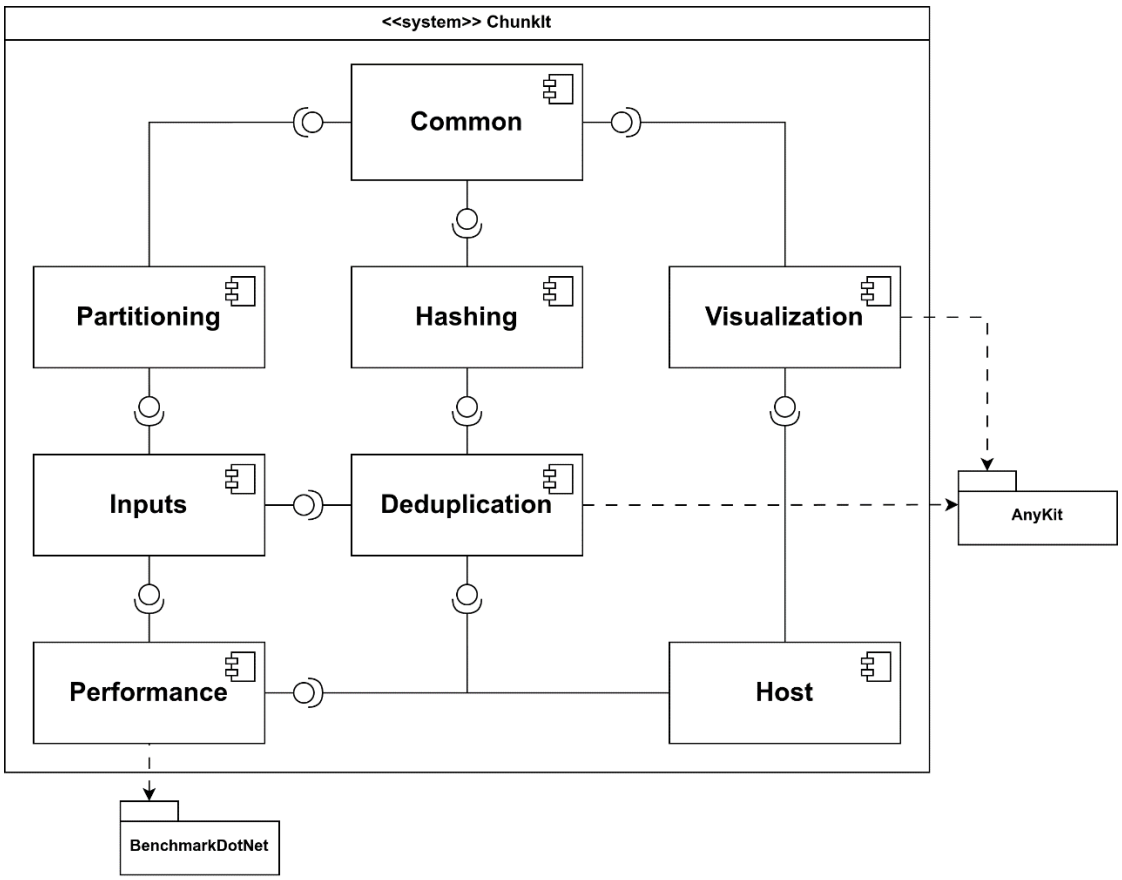


Рис. 12. Діаграма модулів програмного комплексу *ChunkIt*

Базовий рівень архітектури складає модуль *Common*, що містить спільні для решти модулів абстракції, зокрема класи та інтерфейси, що формалізують контракти для реалізацій методів фрагментації, хешування та збору метрик.

Модуль *Hashing* інкапсулює реалізації алгоритмів хешування даних, які використовуються для розрахунку значень хешу вмісту фрагментів даних.

Модуль *Partitioning* містить реалізації всіх розглянутих методів фрагментації даних, включно з модифікованим методом Twin CDC.

Конфігураційний аспект реалізовано модулем *Inputs*, що описує вхідні параметри для виконання бенчмарків, тобто формує контексти збору метрик шляхом декартового добутку множини всіх методів фрагментації з проєкту *Partitioning* та множини вхідних файлів.

Збір та розрахунок метрик реалізовано у модулях *Performance* та *Deduplication*. Проєкт *Performance* відповідає за зняття метрик

продуктивності, інкапсулюючи всі необхідні залежності та налаштування середовища для відтворюваного вимірювання пропускної здатності фрагментації даних. В свою чергу, проєкт *Deduplication* реалізує конвеєр розрахунку метрик дедублікації. Обидва проєкти отримують заздалегідь підготовлену конфігурацію, задану модулем *Inputs*, та виконують методи з модулів *Partitioning* та *Hashing*, формуючи звіти про метрики продуктивності та дедублікації, що надалі використовуються модулем візуалізації.

Модуль візуалізації метрик *Visualization* містить необхідні залежності для роботи з графічними представленнями та реалізує конвеєр візуалізації звітів про продуктивність та дедублікацію. Модуль формує різноманітні графічні матеріали, базуючись на даних, що були сформовані модулями збору метрик, однак не взаємодіє безпосередньо з методами фрагментації.

Модуль *Host* виконує роль вхідної точки застосунку та оркестратора всього процесу збору метрик та їх візуалізації. Він послідовно ініціалізує конфігурації експериментів, запускає конвеєри зняття метрик продуктивності та дедублікації, агрегує результати й передає їх до модуля візуалізації.

Узагальнюючи, архітектура програмного комплексу *ChunkIt* забезпечує чітке розділення відповідальностей між модулями фрагментації, хешування, збору метрик та їх візуалізації. Це забезпечує гнучкість програмного забезпечення, завдяки чому інтеграція нових методів фрагментації та метрик не викликатиме складнощів, а також гарантує відтворюваність та прозорість усього експериментального конвеєра.

3.5. Абстракції фрагментації даних

3.5.1. Chunk

Структура *Chunk* репрезентує мінімальну одиницю фрагментованих даних, тобто єдиний незалежний фрагмент. Як зображено

лістингом 5 (додаток 2), кожен фрагмент даних є незмінним та має всього чотири властивості.

Послідовний ідентифікатор фрагменту (Id; k). Кожен фрагмент даних має унікальний послідовний номер. Дана властивість не бере прямої участі у індексації та відновленні файлів, однак може бути використана для налагодження методів фрагментації даних або індексу (рецепту або сховища) даних, оскільки послідовні значення простіше сприймаються людиною у порівнянні з глобальними зсувами.

Глобальний зсув фрагменту (Offset; o_k). Глобальний зсув фрагменту даних фізично визначає місце розташування фрагменту даних у файлі, тобто починаючи з якого байту починається цей фрагмент. Це одна з ключових властивостей фрагментів даних, оскільки вона використовується для адресації даних, а також є індикатором цілісності даних. Очевидно, що глобальний зсув поточного фрагменту має бути рівним сумі розмірів усіх попередніх фрагментів, тому таким чином можна впровадити перевірку цілісності даних під час відновлення файлу з індексу (рецепту).

Розмір фрагменту (Length; l_k). Кожен фрагмент також характеризується розміром (довжиною) даних, що міститься у ньому. Дана властивість також бере участь у перевірці цілісності даних, як це було описано раніше.

Хеш фрагменту (Hash, h_k). Хеш фрагменту вираховується на основі даних, що містяться у цьому фрагменті, при чому застосована хеш функція має бути криптографічно стійкою, тобто мати нескінченно малу ймовірність колізії значень хешів для різних даних. Кожен утворений фрагмент даних ідентифікується саме за цією властивістю, тобто два фрагменти вважаються еквівалентними за вмістом якщо хеші порівнюваних фрагментів рівні.

3.5.2. ChunkReader

Клас *ChunkReader* призначений для зчитування фрагментів даних з вихідного потоку даних (*Stream*). Даний клас інкапсулює послідовне

розбиття довільного потоку байтів на контенто-залежні фрагменти та, за потреби, обчислення їхніх хешів. *ChunkReader* параметризується методом фрагментації даних хешування утворених фрагментів даних шляхом використанням спільного програмного інтерфейсу. Таким чином, *ChunkReader* поєднує у собі фрагментацію та хешування, при цьому не маючи залежності від конкретних методів фрагментації чи хешування. Даний клас виконує дві основні задачі.

Наповнення буферу даних. На кожній ітерації утворення нового фрагменту даних, *ChunkReader* має наповнити вхідний буфер даних, що ділиться на дві частини: буфер залишку даних з минулої ітерації та буфер зчитування з вихідного потоку даних. Після утворення фрагменту даних, цілком природно мати певний залишок даних, що не потрапив в останній фрагмент даних та має бути опрацьованим у наступній ітерації. Таким чином, залишок даних копіюється на початок вхідного буферу, а решта цього буферу заповнюється новими даними з вихідного потоку. На рис. 13 зображено логічне розбиття вхідного буферу на залишковий буфер та буфер зчитування. Варто зауважити, що у випадку, коли залишковий буфер має розмір більший за максимальний розмір фрагменту S_{max}, операцій копіювання та зчитування не відбувається, адже заздалегідь відомо, що будь-який з методів фрагментації даних не може утворити фрагмент більший за S_{max}, а тому необхідність наповнювати вхідний буфер даних відпадає; в цьому випадку роль вхідного буферу даних відіграє залишковий буфер.

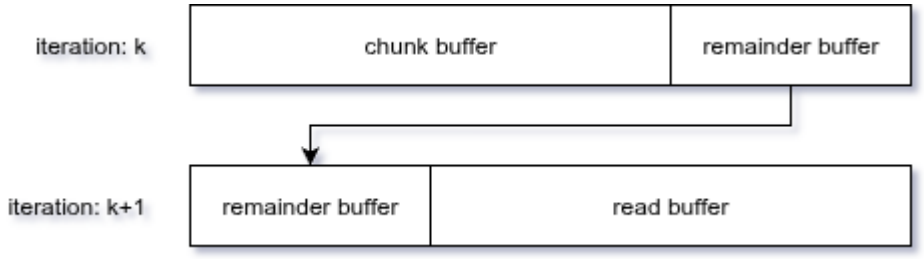


Рис. 13. Ділення вхідного буферу на залишковий буфер та буфер зчитування

Опціональна емісія фрагментів. У специфічних сценаріях фрагментації даних, в яких відсутня необхідність утворення фрагментів даних, утворення фрагментів делегується іншому компоненту або з метою уникнення накладних витрат на емісію фрагментів даних, клас *ChunkReader* надає два публічні методи для фрагментації даних: *ReadChunksAsync* та *ReadChunkLengthsAsync*. Перший з наведених методів реалізує повний механізм фрагментації даних, включаючи наповнення вхідного буферу, пошук меж, хешування та емісію фрагментів. В свою чергу, другий метод представляє собою спрощену версію механізму фрагментації, що лише наповнює вхідний буфер та повертає глобальні зсуви фрагментів даних. Таким чином, наприклад, у сценарії виміру метрик продуктивності методів фрагментації даних отримані метрики не будуть включати накладні витрати на емісію фрагментів даних.

Асинхронне перерахування фрагментів. Оскільки зчитування вхідного потоку даних може вимагати виконання блокуючих I/O операцій, клас *ChunkReader* повністю підтримує асинхронне зчитування потоку байтів. Асинхронне зчитування дозволяє не блокувати потік виконання інструкцій, а навпаки, звільнити поточний потік виконання на час виконання I/O операції, тим самим не витрачаючи ресурс CPU на очікування. Крім того, обидва публічні методи класу *ChunkReader* надають інтерфейс потокового перерахування фрагментів (*IAsyncEnumerable<Chunk>*) або їх розмірів (*IAsyncEnumerable<int>*), за якого кожний фрагмент або його розмір повертається споживачеві одразу після визначення межі фрагменту. Такий режим забезпечує поелементну обробку даних зовнішнім споживачем по мірі утворення фрагментів, усуваючи потребу в попередній матеріалізації повної послідовності фрагментів, при цьому зменшуючи латентність і пікове споживання пам’яті.

3.5.3. GearTable

Клас GearTable реалізує класичний ковзний хеш Gear і слугує низькорівневою безалокаційною примітивою для побайтового оновлення ковзного хешу. Даний клас інкапсулює 256-елементну таблицю (по одному значенню на кожне можливе значення типу *byte*) та надає публічний метод *Fingerprint*, який оновлює значення ковзного хешу згідно (25) та лістингу 2 (додаток 2). Даний метод мінімізує копіювання значень аргументів шляхом отримання цих аргументів за посиланнями, що дає можливість JIT компілятору виконати агресивний method inlining у вимогливій ділянці інструкцій. Зазначені оптимізації забезпечують сталу амортизовану вартість розрахунку хеша рівною O(1) на байт. Окремі екземпляри класу *GearTable* можуть застосовуватися для розрахунку ковзних хешів у лівому і правому вікнах незалежно за умови двонаправленого сканування, що застосовується в модифікованому методі Twin CDC. Для створення екземплярів класу *GearTable* доступні два фабричні методи.

Random. Перший фабричний метод *Random* заповнює таблицю Gear випадковими значеннями використовуючи алгоритм генерації псевдовипадкових значень SplitMix [29]. У цьому випадку детермінізм утворення таблиць Gear зберігається завдяки параметризованому кореню генерації випадкових чисел (так званий *seed*). За умови використання одного й того ж самого значення *seed*, окремо згенеровані Gear таблиці міститимуть ідентичні значення, тобто такі таблиці будуть поелементно рівні.

Predefined. Другий фабричний метод *Predefined* використовує заздалегідь підготовлені значення таблиці Gear, що були запозичені з репозиторію DedupBench [30]. Крім наведених значень, даний метод дозволяє виконувати циклічну ротацію байтів кожного зі значень таблиці, таким чином утворюючи 64 можливих варіанти таблиці на базі одного набору значень.

3.5.4. Hashing

Для хешування вмісту фрагментів використовується спільний програмний інтерфейс, що дозволяє приховати конкретні методи хешування шляхом застосування абстракції *IHasher*, що наведена у лістингу 6 (додаток 2). Даний інтерфейс інкапсулює операцію обчислення детермінованого хешу для байтової послідовності довільної довжини та повертає ідентифікатор фрагмента фіксованої довжини, який використовується в якості ідентифікатора вмісту фрагменту в індексі.

В залежності від вимог до стійкості колізій та продуктивності, можуть бути використані різні алгоритми хешування фрагментів даних. Так, наприклад, для надійної ідентифікації вмісту фрагментів даних необхідно застосувати криптографічно стійкі алгоритми (SHA-256, SHA-512, Blake3). Проте, якщо колізії хешів вмісту даних допустимі, в такому разі перевага надається більш простим та швидким алгоритмам (MD5, Xx128). Всі наведені алгоритми хешування вмісту даних реалізовані з використанням спільного програмного інтерфейсу, що зображено у лістингу 6 (додаток 2). Крім того, було також реалізовано функцію хешування, яка завжди повертає пустий хеш. Ця реалізація необхідна у випадках, коли значення хешів вмісту фрагментів не використовуються, а отже відпадає необхідність витрачати ресурси та час на розрахунок хешів кожного утвореного фрагменту.

3.6. Збір та обробка метрик контенто-залежної фрагментації даних

З метою компонування послідовності етапів отримання та обробки метрик контенто-залежної фрагментації даних, було розроблено конвеєри обробки даних, що виконуються один за одним для кожної заданої пари методу фрагментації та вхідного файлу, що надалі будемо називати контекстом збору метрик фрагментації даних.

3.6.1. Збір метрик продуктивності

Для отримання метрик продуктивності розглянутих та модифікованого методів контенто-залежної фрагментації даних було

імплементовано бенчмарк продуктивності методів фрагментації даних. Бенчмарк реалізовано одним параметризованим методом, в тілі якого виконується читання розмірів фрагментів даних за допомогою методу *ReadChunkLengthsAsync*. Оскільки цей метод не виконує емісії фрагментів даних та лише перераховує розміри утворених фрагментів даних, отримані метрики свідчитимуть виключно про про продуктивність обраних методів.

Бенчмарк продуктивності параметризується контекстом збору метрик фрагментації даних, що включає в себе метод фрагментації даних та вхідний файл.

Метод фрагментації даних. Для порівняння продуктивності різних методів фрагментації даних необхідно виконати наведений бенчмарк фрагментації з усіма обраними та модифікованим методами фрагментації даних.

Вхідний файл. Оскільки різні методи фрагментації мають власні предикати встановлення меж фрагментів, результат фрагментації даних буде відрізнятися між будь-якими обраними методами. Отже, метрики будь-якого обраного методу можуть відрізнятися на різних вихідних файлах. Тому для виміру продуктивності обраних та модифікованого методів фрагментації даних пропонується обрати декілька вихідних файлів з різними характеристиками розміру, ентропії байтів та очікуваної повторюваності фрагментів даних.

Як вже було зазначено раніше, метою бенчмарку є вимірювання метрик продуктивності саме методів фрагментації даних, а тому всі інструкції, що не стосуються процесу фрагментації, мають бути відокремлені від інструкцій бенчмарку.

Ініціалізація бенчмарку (global setup). На етапі ініціалізації бенчмарку необхідно підготувати вихідний потік байтів поточного вихідного файлу (*FileStream*). Вихідний потік створюється виключно для вже існуючого на файловій системі файлу (*FileMode.Open*). Файл відкривається лише для читання (*FileAccess.Read*), при чому доступ до цього файлу має бути

ексклюзивним (*FileShare.None*). Розмір внутрішнього буферу зчитування потоку байтів задається рівним 4 мегабайтам. Також, оскільки заздалегідь відомо, що всі методи фрагментації даних працюють з обмеженими послідовними буферами даних розміром S_{max}, тобто читання вихідного потоку байтів є послідовним (включаючи Twin CDC, оскільки двонаправлений пошук виконується в межах поточного буферу даних), для збільшення продуктивності читання даних з файлової системи задається опція послідовного читання файлу (*FileOptions.SequentialScan*), що дозволяє ядру операційної системи застосувати більш ефективне кешування порцій файлу [31].

Також, крім ініціалізації вихідного потоку байтів, необхідно також створити екземпляр класу *ChunkReader*, використовуючи який і буде відбуватися читання вихідного потоку байтів. Це можливо зробити всього один раз на етапі ініціалізації бенчмарку, оскільки *ChunkReader* є класом, що не містить стану (stateless class). Стан процесу фрагментації даних обмежується контекстом виклику методів *ReadChunksAsync* та *ReadChunkLengthsAsync*.

Ініціалізація ітерації (iteration setup). До виконання першої ітерації бенчмарку, внутрішній вказівник потоку даних знаходиться на початку цього потоку. Проте, після виконання першої ітерації бенчмарку цей вказівник вже буде знаходитися в кінці потоку, адже бенчмарк викликає метод *ReadChunkLengthsAsync*, аргументом якого є створений потік даних на етапі ініціалізації бенчмарку. Отже, всі наступні ітерації бенчмарку опрацьовуватимуть потік даних, що вже було вичерпано, а тому результати такого бенчмарку будуть не репрезентативні. Для того, щоб уникнути наведеного спотворення метрик продуктивності, необхідно повертати внутрішній вказівник потоку даних на початок перед виконанням кожної ітерації бенчмарку. Це можливо завдяки методу ініціалізації ітерації, де і виконується необхідне скидання вказівника потоку даних.

Звільнення ресурсів бенчмарку (global cleanup). По закінченню всіх ітерацій бенчмарку необхідно звільнити ресурси, що були використані цим бенчмарком. Тому, файловий потік, що був створений на етапі ініціалізації бенчмарку, має бути утилізований. Під час утилізації об’єкту потоку даних, ексклюзивний доступ до відповідного файлу у файловій системі втрачається, отже, файл знову стає доступним іншим споживачам, наприклад, наступній ітерації бенчмарку.

Результатом збору метрик продуктивності є звіт про продуктивність фрагментації даних, що включає в себе:

- середній час виконання фрагментації даних;
- розраховане значення пропускної здатності фрагментації.

3.6.2. Збір метрик дедублікації

Для збору метрик дедублікації даних було створено вкладений конвеєр, збору метрик дедублікації для кожного з контекстів збору метрик фрагментації даних. Даний конвеєр складається з п’яти етапів, а результатом його виконання є звіт дедублікації даних, що містить метрики дедублікації, розраховані на основі утворених фрагментів.

Етап 1. Фрагментація даних. На першому етапі відкривається вхідний потік даних та виконується фрагментація даних цього потоку заданим методом. Результатом виконання цього етапу є впорядкований набір утворених фрагментів даних.

Етап 2. Валідація фрагментів даних. Щоб переконатися у коректності утвореної послідовності фрагментів даних, на цьому етапі виконується розрахунок сумарного розміру утворених фрагментів даних та перевірка умови збереження розміру файлу згідно (9). В разі, якщо дана умова не виконується, отриманий звіт про дедублікацію слід вважати помилковим, тому подальші етапи конвеєру не будуть виконані.

Етап 3. Розрахунок середнього розміру фрагментів. На цьому етапі розраховується середнє арифметичне значення розміру утворених фрагментів.

Етап 4. Розрахунок коефіцієнту дедублікації даних. Згідно (5), для визначення фактичного збереженого дискового простору розраховується коефіцієнт дедублікації даних без врахування розміру індексу. На відміну від бенчмарків продуктивності, де хеш вмісту фрагментів даних не має значення, а тому використовується заглушка алгоритму хешування вмісту фрагментів, під час розрахунку метрик дедублікації даних, зокрема для розрахунку коефіцієнту дедублікації, необхідно використовувати криптографічно стійку хеш-функцію. В даному дослідженні було використано SHA-256, проте можна використати будь-яку іншу криптографічно стійку хеш-функцію.

Етап 5. Розрахунок геометричного коефіцієнту якості фрагментації даних. Згідно (15) та (16) розраховується геометричний коефіцієнт якості фрагментації даних.

Отже, результатом виконання конвеєру збору метрик дедублікації для кожного контексту збору метрик є звіт про дедублікацію даних, що включає в себе:

- послідовність утворених фрагментів;
- середній розмір фрагментів;
- обсяг дедублікованих фрагментів;
- коефіцієнт дедублікації даних (D);
- коефіцієнт девіації фрагментів даних (V);
- геометричний коефіцієнт якості фрагментації даних (Q).

3.6.3. Зведені метрики

Для розрахунку метрики пропускної здатності дедублікації даних необхідно попередньо отримати обидва звіти про продуктивність та дедублікацію, оскільки розрахунок цієї метрики, відповідно до (12), вимагає

наявність значень коефіцієнту дедублікації та часу виконання фрагментації даних. Тому, після збору звітів про продуктивність та дедублікацію, також виконується групування отриманих звітів за контекстом збору метрик фрагментації даних, а для кожної з утворених груп розраховується зведена метрика пропускної здатності дедублікації даних.

3.6.4. Візуалізація метрик

Для подальшого аналізу та порівняння метрик фрагментації даних було реалізовано окремий конвеєр візуалізації, що обробляє звіти, сформовані конвеєром збору метрик продуктивності та дедублікації. Кожен етап цього конвеєра відображає одну з обраних метрик фрагментації, причому значення метрик подано в агрегованому вигляді для кожного вхідного файлу, що забезпечує можливість зручного порівняння значень метрик різних методів фрагментації на одному файлі.

Крім того, для підвищення наочності порівняння та спрощення візуальної ідентифікації, кожному методу фрагментації даних було призначено сталий колір на всіх візуалізаціях. Завдяки цьому, значення будь-якої з метрик одного й того самого методу легко відстежувати та порівнювати відносно різних вхідних файлів.

Етап 1. Візуалізація пропускної здатності фрагментації даних. На першому етапі конвеєра здійснюється візуалізація метрики пропускної здатності фрагментації даних, обчисленої за результатами виконання бенчмарку продуктивності для кожного контексту збору метрик. Значення пропускної здатності подаються у вигляді гістограми, де вісь X відповідає методу фрагментації даних, а вісь Y – обсягу даних, оброблених за одиницю часу. Для кожного вхідного файлу формується окрема гістограма, що дає змогу безпосередньо порівнювати продуктивність різних методів фрагментації за швидкодією на одному наборі даних.

Етап 2. Візуалізація пропускної здатності дедублікації даних. Другий етап конвеєра призначений для візуалізації пропускної здатності

дедублікації даних. За отриманими метриками продуктивності для кожного контексту збору метрик формується гістограма, де по осі X відкладаються методи фрагментації, а по осі Y – пропускна здатність дедублікації. Аналогічно до попереднього етапу, значення згруповано за вхідними файлами, що дозволяє оцінити, як зміна методу фрагментації впливає на продуктивність дедублікації для одного й того ж файлу.

Етап 3. Візуалізація коефіцієнта дедублікації даних. На третьому етапі виконується візуалізація коефіцієнта дедублікації даних, що характеризує відносний обсяг збереженого дискового простору в сховищі за рахунок усунення дубльованих фрагментів. Коефіцієнти дедублікації для всіх звітів фрагментації даних відображаються на спільній гістограмі, де кожен метод фрагментації представлено окремим стовпчиком. Вісь X відповідає методу фрагментації, а вісь Y – значенню коефіцієнта дедублікації.

Етап 4. Візуалізація коефіцієнта девіації фрагментів даних. Четвертий етап спрямований на візуальне порівняння коефіцієнту девіації фрагментів даних відносно цільового середнього розміру. Отримані значення цього коефіцієнту також візуалізуються на спільній гістограмі, де вісь X представляє метод фрагментації, вісь Y – значення коефіцієнта девіації. Такий підхід дозволяє порівняти розподіл розмірів фрагментів даних, що утворюються різними методами на одних і тих самих вхідних файлах.

Етап 5. Візуалізація геометричного коефіцієнта якості фрагментації даних. П’ятий етап конвеєра узагальнює результати попередніх метрик шляхом візуалізації геометричного коефіцієнта якості фрагментації даних, який інтегрально відображає баланс між ефективністю дедублікації та якістю розподілу фрагментів. Аналогічно до попереднім візуалізаціям, значення геометричного коефіцієнта якості для кожного звіту фрагментації подано на спільній гістограмі, де вісь X відповідає методу фрагментації, а вісь Y – значенню геометричного коефіцієнта якості

фрагментації даних. Така форма подання дозволяє оцінити кожен метод з точки зору зведеної метрики якості фрагментації даних.

3.7. Висновки до розділу 3

В контексті даного дослідження було розроблено програмний комплекс, що реалізує розглянуті та модифікований методи фрагментації даних. Крім того, розроблюваний комплекс також дозволяє оцінити та порівняти кількісні характеристики цих методів.

Для програмної реалізації проаналізованих та модифікованого методу контенто-залежної фрагментації даних було обрано мову C# 13.0 на платформі .NET 9. Такий вибір зумовлений поєднанням високорівневої безпеки пам’яті й зрілих засобів для написання високопродуктивного коду, зокрема сучасного JIT-компілятора RyuJIT та інструментів для роботи з пам’яттю. У сукупності це дозволяє створювати продуктивний, кросплатформенний та підтримуваний код, зокрема для інтенсивних операцій над потоками байтів, характерних для CDC методів.

4. ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТРИК МОДИФІКОВАНОГО ТА РОЗГЛЯНУТИХ МЕТОДІВ КОНТЕНТО-ЗАЛЕЖНОЇ ФРАГМЕНТАЦІЇ ДАНИХ

4.1. Тестові набори даних

Для проведення порівняльного аналізу методів контенто-залежної фрагментації даних було сформовано чотири тестові набори даних, що відображають як реальні сценарії використання дедублікації, так і наближений до найгіршого випадок для методів CDC [32].

Перший тестовий набір даних *linux kernel* являє собою архів вихідного коду ядра Linux, сформований шляхом завантаження вихідного коду з офіційного CDN у вигляді стиснутого архіву, подальшої декомпресії та перепакування у архів *tar* без застосування стиснення. Набір даних переважно містить код мовою програмування C та складається з останніх 10 релізів, а загальний обсяг архіву становить 15501776 байтів (15.87 гігабайт або 126.99 гігабіт).

Другий тестовий набір *dotnet runtime* утворений аналогічно до першого. Даний набір даних містить код різними мовами програмування, переважно C# та C++. До складу даного набору даних входять останні 15 релізів, а загальний обсяг становить 13681387520 байтів (13.68 гігабайт або 109.45 гігабіт).

Третій тестовий набір *gcc* створений аналогічно до перших двох та містить вихідний код мовами C++ та Ada. Даний набір даних містить останні 23 релізи, а його загальний обсяг становить 17321676800 байтів (17.32 гігабайт або 138.57 гігабіт).

Четвертий та останній набір даних *random* було утворено синтетично за допомогою стандартного пристрою операційної системи Linux */dev/urandom*. Висока ентропія такого потоку даних на практиці унеможливлює виявлення повторюваних фрагментів. Цей набір даних використовується передусім для оцінювання пропускної здатності методів

CDC на високоентропійних випадкових даних, що є найгіршим випадком у процесі дедублікації. Обсяг цього набору даних складає 5368709120 байтів (5.37 гігабайт або 42.95 гігабіт).

4.2. Хост виконання бенчмарків

Для проведення ресурсомістких бенчмарків продуктивності було використано хост на базі процесора AMD Ryzen AI 9 HX 370, оснащений 32 гігабайтами оперативної пам’яті DDR5 та твердотільним накопичувачем WD PC SN740. В якості операційної системи було використано *Arch Linux* із модифікованим ядром *linux-zen* версії 6.16.5.

Збірку програмного комплексу було виконано у конфігурації *Release*, за якої застосовуються низькорівневі оптимізації на етапі компіляції програмного коду C# у Common Intermediate Language. Також варто зазначити, що за цієї конфігурації результуюча збірка не міститиме засобів налагодження, а отже і накладні витрати ресурсів на підтримку налагодження також будуть відсутні.

Виконання даного програмного комплексу здійснювалися з використанням платформи .NET версії 9.0.301, що включає оптимізації часу виконання *Dynamic PGO* та *Tiered Compilation*.

Даний хост було інстальовано та налаштовано окремо від будь-яких користувацьких процесів та інтерфейсів за принципом *bare metal* – без віртуалізації, супервайзора, та інших сторонніх сутностей, що могли негативно вплинути на результати бенчмарків. Для необмеженого доступу до апаратних ресурсів, бібліотека виконання бенчмарків BenchmarkDotNet вимагає, щоб запуск програмного комплексу відбувався від імені адміністратора, а саме *root* в операційній системі на базі ядра *linux*. Таким чином, кожен з бенчмарків запускається з максимально можливим пріоритетом, тобто значення *nice level* рівне *-20*.

4.3. Параметризація методів

Всі методи контенто-залежної фрагментації мають один чи декілька параметрів, згідно яких виконується фрагментація даних, при чому деякі параметри є характерними лише одному конкретному методу. З метою отримання відновлюваних результатів бенчмарків продуктивності та дедублікації, кожен з таких параметрів було зафіксовано.

Спільним для всіх методів є параметр цільового середнього розміру фрагментів даних S_{conf}, тож кожен з обраних методів контенто-залежної фрагментації даних було сконфігуровано значеннями, що наведені у табл. 1.

Таблиця 1

Значення цільового розміру фрагментів S_{conf}

Компонент цільового розміру фрагментів даних S_{conf}	Значення компонента (кілобайт)
Мінімальний розмір фрагментів (S_{min})	8
Середній розмір фрагментів (S_{avg})	16
Максимальний розмір фрагментів (S_{max})	32

У табл. 2 наведено значення параметрів, що характерні для кожного індивідуального методу CDC. Варто зазначити, що базовий метод Fast CDC та обидві варіації модифікованого методу Twin CDC, а саме *twin-mono* та *twin-duo*, використовують однакове значення параметру рівня нормалізації (normalization_level) для коректного порівняння, при чому *twin-mono* використовує одну Gear-таблицю, а *twin-duo* – дві Gear-таблиці.

Таблиця 2

Значення індивідуальних параметрів методів

Метод CDC	Назва параметру	Значення параметру
fast	*normalization_level*	3
ram	*window_size*	7936
ae	*window_size*	7936
seq-incr	*mode*	*increasing*
	sequence_length	5
	skip_trigger	50
	skip_length	512
twin-mono	*normalization_level*	3
	gear_tables	1
twin-duo	*normalization_level*	3
	gear_tables	2

4.4. Порівняння метрик дедублікації

4.4.1. Коефіцієнт дедублікації даних

Коефіцієнт дедублікації даних *D* є базовою кількісною характеристикою ефективності процесу дедублікації та визначається як відношення обсягу дедублікованих даних до загального початкового обсягу даних. В рамках проведених бенчмарків дедублікації, розрахунок значень коефіцієнту *D* було впроваджено згідно (5), тобто було знехтувано розміром

утворених індексів фрагментів, оскільки їх обсяг складав менше 0.1% від обсягу вхідних потоків даних.

На рис. 14 зображено порівняльний графік значень коефіцієнтів дедублікації даних обраних методів контенто-залежної фрагментації даних за результатами виконання фрагментації обраних наборів даних.

Очевидно, що метод Seq CDC має найвище значення коефіцієнту *D* на наборах *linux*, *dotnet* та *gcc*, однак, як буде показано далі, такий результат було отримано лише за рахунок великої кількості аномально малих фрагментів даних.

Без врахування Seq CDC, метод Rabin має кращий результат на 1% на наборі *gcc*. На наборах *linux* та *dotnet*, краще значення коефіцієнту *D* має модифікований метод Twin CDC, а перевага складає 2-3%.

Як і очікувалося, для синтетичного високоентропійного набору даних *random* значення коефіцієнта дедублікації всіх обраних методів фрагментації наближуються рівні нулю, що відповідає практичній відсутності повторюваних фрагментів у даних і відображає наближений до найгіршого випадок для систем дедублікації.

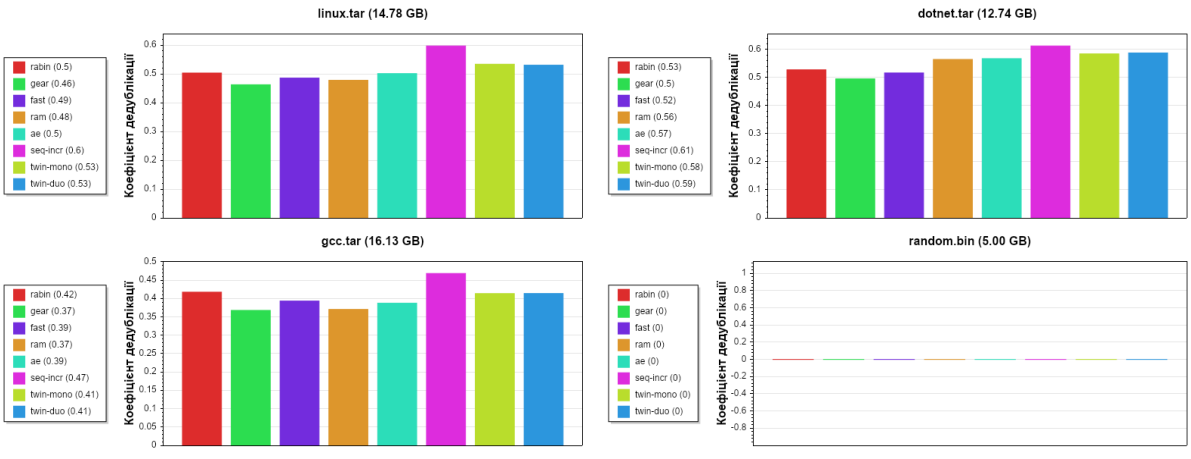


Рис. 14. Порівняльний графік коефіцієнтів дедублікації даних

4.4.2. Коефіцієнт девіації фрагментів даних

Коефіцієнт девіації фрагментів даних характеризує ступінь відхилення фактичного розподілу розмірів фрагментів даних від цільового

середнього розміру S_{avg}. Значення коефіцієнта ближчі до одиниці відповідають більш вузькому розподілу, у якому більшість фрагментів зосереджена поблизу S_{avg}, а частка як надто малих, так і надто великих фрагментів є мінімальною. Розрахунок значень коефіцієнту девіації V було впроваджено відповідно до (15).

На рис. 15 зображено порівняльний графік значень коефіцієнтів V. Як вже було згадано у порівнянні значень коефіцієнту дедублікації D, метод Seq CDC має гірше значення коефіцієнт девіації на всіх наборах даних, а з врахування значень обох коефіцієнтів $D_{seq\text{-}incr}$ та $V_{seq\text{-}incr}$ випливає, що методом Seq CDC досягнуто високого рівня дедублікації шляхом утворення великої кількості аномально малих фрагментів даних, що позитивно впливає на ймовірність знаходження повторюваних фрагментів, при цьому розмір індексу фрагментованих даних зростає, а продуктивність відновлення даних з індексу спадає.

На наборі даних *gcc*, модифікований метод Twin CDC має значення коефіцієнту $V_{twin\text{-}mono}$ вище на 1% відносно наступного кращого значення V_{ae}. На наборах *linux*, *dotnet* та *random*, метод Twin CDC поступається методам AE та RAM на 6-7%. Варто зауважити, що Twin CDC має рівне або краще значення коефіцієнту V відносно базового методу Fast CDC та батьківських методів Rabin та Gear, яким характерне утворення аномально великих фрагментів даних.

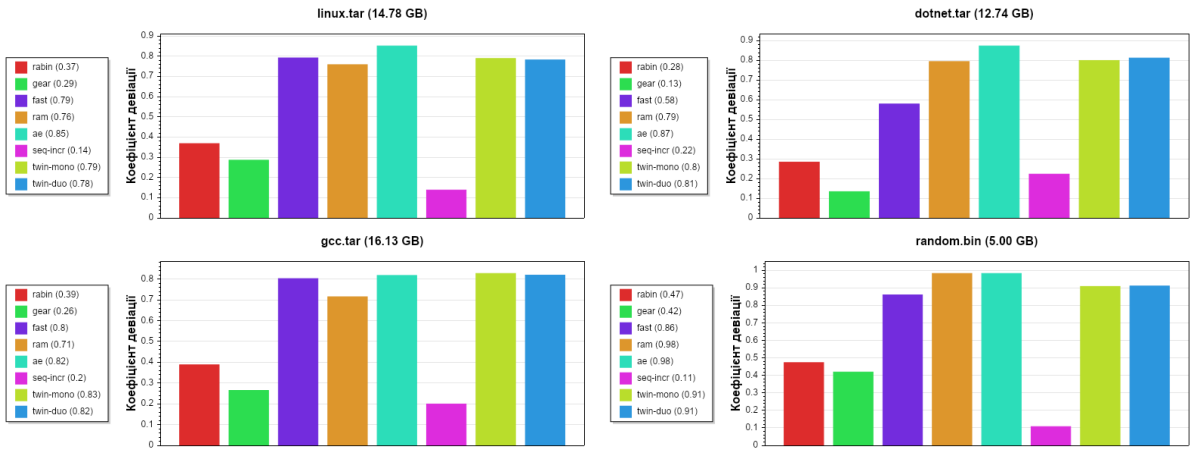


Рис. 15. Порівняльний графік коефіцієнтів девіації фрагментів даних

4.4.3. Геометричний коефіцієнт якості фрагментації даних

Геометричний коефіцієнт якості фрагментації даних є інтегральною метрикою, що узагальнює коефіцієнти дедублікації *D* та девіації *V*. Як вже було емпірично показано на методі Seq CDC, за утворення великої кількості аномально малих фрагментів, тобто зниження коефіцієнту девіації *V*, збільшується обсяг дедублікованих даних, тобто зростає коефіцієнт дедублікації *D*. З метою демонстрації балансу значень коефіцієнтів *D* та *V*, було введено узагальнений геометричний коефіцієнт *Q*, розрахунок якого впроваджено згідно з (16).

З порівняльного графіку, зображеного на рис. 16, видно, що методи Rabin, Gear та Seq CDC мають помірні значення коефіцієнта Q через низьке значення коефіцієнту *V*.

На наборі *gcc*, модифікований метод Twin CDC має вище значення коефіцієнту *Q* на 2%, а на наборі *linux* значення $Q_{twin\text{-}mono}$ рівне значенню Q_{ae}. На наборі *dotnet* найкращим значенням є Q_{ae} з перевагою в 1% відносно Twin CDC.

Очевидно, що через неможливість дедублікації високоентропійного набору даних *random*, значення *Q* всіх методів контенто-залежної фрагментації даних рівне нулю, оскільки значення коефіцієнту *D* також рівне нулю.

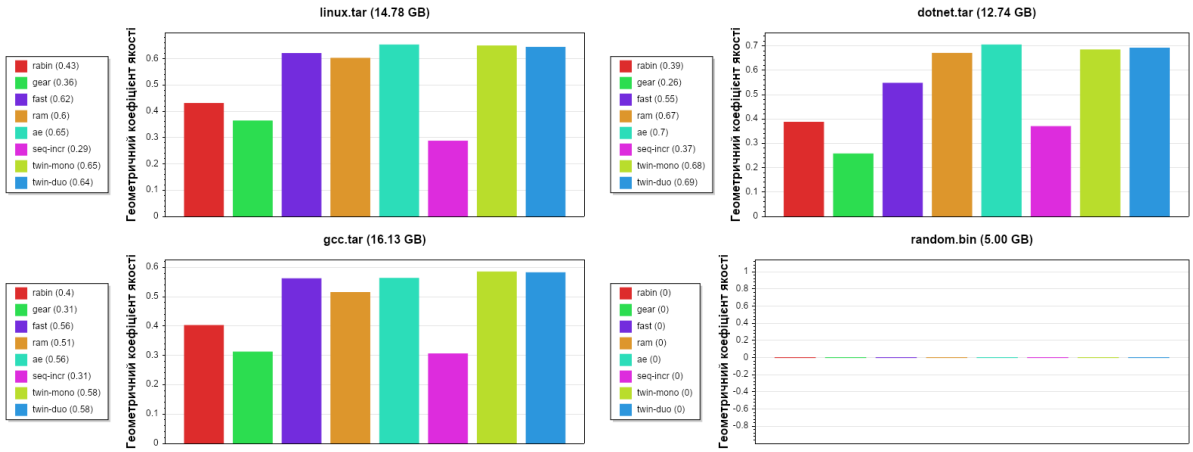


Рис. 16. Порівняльний графік геометричних коефіцієнтів якості фрагментації даних

4.4.4. Висновки до порівняння метрик дедублікації

Завдяки реалізованим та виконаним бенчмаркам дедублікації даних, а також після проведеного порівняльного аналізу метрик дедублікації, можна зробити висновок, що модифікований метод Twin CDC має вищий коефіцієнт дедублікації даних *D* на 2-3% на двох з трьох наборах даних, при цьому зберігши конкурентний рівень значень коефіцієнтів девіації фрагментів *V* якості фрагментації *Q*, маючи значення цих коефіцієнтів вищі, рівні або з невеликим відставанням від наступного кращого лідера, в залежності від набору даних.

4.5. Порівняння метрик продуктивності

4.5.1. Пропускна здатність фрагментації даних

Пропускна здатність контенто-залежної фрагментації даних розглядається як відношення обсягу вхідного файлу до часу роботи методу фрагментації та вимірюється у гігабітах на секунду. Усі заміри виконувалися лише для етапу контенто-залежної фрагментації, без урахування витрат на обчислення хешів, роботу з індексом фрагментів та інші складові конвеєра дедублікації. Таким чином, отримані значення метрики пропускної здатності характеризують виключно швидкодію методів без урахування накладних витрат на інші етапи дедублікації даних. Розрахунок пропускної здатності фрагментації даних було впроваджено згідно з (11).

На рис. 17 зображено порівняльний графік, де наведені значення пропускної здатності фрагментації даних обраних методів на чотирьох тестових наборах даних. На кожному з наборів даних метод Rabin демонструє найнижчу пропускну здатність, що пов’язано з обчислювально складним оновленням значення ковзного хеша.

На наборі *linux*, лідером за пропускною здатністю фрагментації є метод Twin CDC (варіація *twin-duo*) зі швидкістю 13.73 гігабіт на секунду та перевагою у 2% відносно наступного кращого Fast CDC.

На наборі *dotnet*, найвищу пропускну здатність мають методи Fast CDC (7.96 гігабіт на секунду) та RAM (7.91 гігабіт на секунду), тоді як метод Twin CDC займає третю позицію (варіація *twin-duo*; 7.85 гігабіт на секунду) та має відставання до 1%.

На наборі *gcc*, очевидним лідером за пропускною здатністю є метод Twin CDC (варіація *twin-duo*; 11.15 гігабіт на секунду), що є кращим показником метрики продуктивності на 39% відносно наступного кращого результату (Fast CDC; 8.04 гігабіт на секунду).

На високоентропійному наборі даних *random*, абсолютну перевагу за продуктивністю мають обидві варіації методу Twin CDC, кращою з яких є *twin-mono* (35.39 гігабіт на секунду) та має перевагу в 249% відносно наступного ліпшого результату (Fast CDC; 14.20 гігабіт на секунду). У цьому випадку, варіація *twin-mono* дає кращий результат за *twin-duo*, оскільки висока ентропія даних забезпечує декореляцію лівого і правого вікон пошуку, а отже і високу ймовірність встановлення межі за умови використання навіть однієї Gear-таблиці, а тому перевага досягається шляхом низькорівневої JIT-оптимізації звернень до єдиного екземпляру Gear-таблиці замість двох окремих.

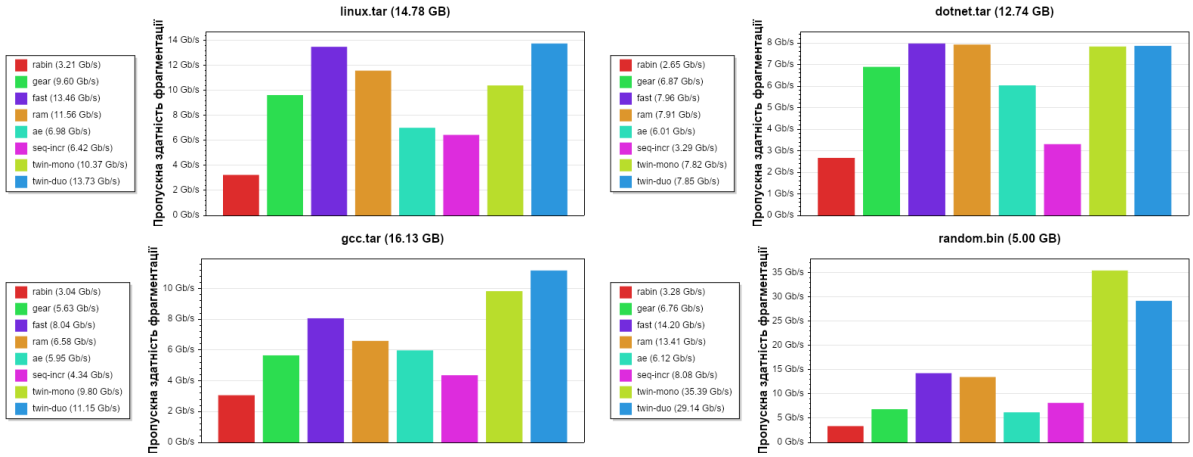


Рис. 17. Порівняльний графік пропускної здатності фрагментації даних

4.5.2. Пропускна здатність дедублікації даних

Під пропускною здатністю дедублікації розуміється ефективна швидкодія конвеєра, що усуває дубліковані фрагменти даних, тобто відношення обсягу усунутих надлишкових даних до часу роботи методу фрагментації даних. На відміну від пропускної здатності фрагментації даних, яка характеризує лише швидкість визначення меж фрагментів, дана метрика одночасно відображає і коефіцієнт дедублікації, і витрати часу на утворення фрагментів. Розрахунок пропускної здатності дедублікації даних було впроваджено згідно з (12).

На рис. 18 зображено порівняльний графік пропускної здатності дедублікації даних, при чому модифікований метод Twin CDC, а саме його варіація *twin-duo*, демонструє вище значення пропускної здатності дедублікації даних, а саме:

- *linux* – на 11 % відносно Fast CDC;
- *dotnet* – на 3% відносно RAM;
- *gcc* – на 46% відносно Fast CDC.

Метрика пропускної здатності дедублікації для набору даних *random* відсутня, оскільки коефіцієнт дедублікації *D* всіх методів фрагментації рівний нулю.

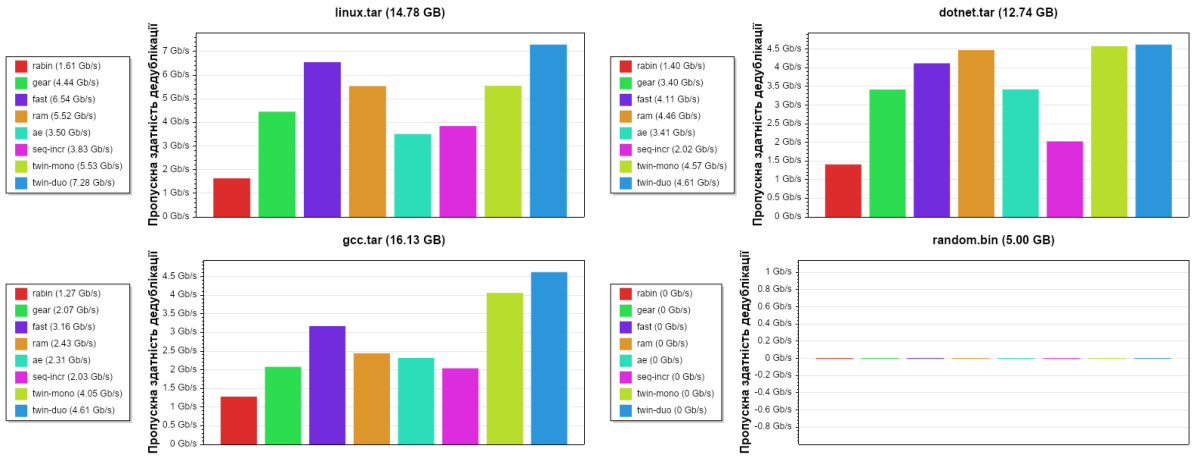


Рис. 18. Порівняльний графік пропускної здатності дедублікації даних

4.5.3. Висновки до порівняння метрик продуктивності

Таким чином, аналіз пропускної здатності модифікованого методу Twin CDC у порівнянні з вже відомими обраними методами контенто-залежної фрагментації показує, що Twin CDC дозволяє істотно збільшити обсяг даних, що фрагментуються та ефективно усуваються за одиницю часу. Отримані результати демонструють приріст пропускної здатності фрагментації від 2% до 39% (без урахування високоентропійного набору даних *random*, на якому перевага складає до 249%), а також приріст пропускної здатності дедублікації від 3% до 46%, в залежності від набору даних.

4.6. Подальші напрямки роботи

4.6.1. Паралелізація двонаправленого пошуку

Поточна реалізація модифікованого методу Twin CDC, як і інші розглянуті методи контенто-залежної фрагментації даних, є однопоточною, тобто вся послідовність пошуку меж фрагментів виконується послідовно. Однак, за наявності двох окремих незалежних вікон пошуку у поточному буфері даних, відкривається можливість паралелізації пошуку меж з використанням двох потоків пошуку, по одному потоку на вікно. Таким чином, потенційно можна досягти ще вищої пропускної здатності без необхідності впровадження додаткових засобів синхронізації потоків виконання завдяки незалежності вікон пошуку.

4.6.2. Оптимізація IO операцій

В межах реалізованого програмного комплексу, зчитування вхідних потоків даних відбувається за допомогою проміжного буферу, що вбудований в стандартні абстракції роботи з вводом-виводом на платформі .NET. Буферизоване зчитування вхідного потоку даних дозволяє зменшити, але не виключити, кількість звернень до файлової системи для зчитування порцій даних. Щобільша кількість таких звернень, тим більша затримка між

ітераціями будь-якого з методів фрагментації даних, тому варто розглянути інші способи зчитування вхідного потоку даних. Так, одним з таких способів може бути попереднє завантаження вхідного потоку даних в оперативну пам’ять, що вимагає виконання IO операцій до фактичної фрагментації даних, однак під час виконання фрагментації затримки звернення до файлової системи відсутні. Варто зауважити, що дана оптимізація може мати різні обмеження в залежності від характеристик хосту, операційної системи та мови програмування. Так, наприклад, платформа .NET дозволяє завантаження потоку даних у пам’ять лише за умови, що обсяг цих даних не перевищує 2 гігабайти [25].

Іншим шляхом оптимізації операцій IO може стати механізм Memory Mapped File, згідно якого вхідний потік даних може бути частково або повністю завантажений у віртуальну пам’ять, а затримки зчитування порцій даних при цьому нижчі, ніж під час зчитування з файлової системи [33].

4.6.3. Реалізація статично-компільованою мовою програмування

Програмний комплекс, що було розроблено в контексті даного, використовує C# на платформі .NET, що є динамічно компільованою мовою програмування, при чому на етапі компіляції C# транслюється у проміжну мову CIL, а на етапі виконання CIL компілюється у машинний код з використанням JIT-компілятора [34]. Використання стандартних абстракцій платформи .NET, а також компіляція на етапі виконання програмного комплексу додають накладних витрат використання апаратних ресурсів, зокрема процесорного часу та обсягу оперативної пам’яті. Крім того, мова програмування C# використовує механізм GC, який також може сповільнювати або навіть призупиняти виконання програмного коду під час звільнення пам’яті від невикористовуваних об’єктів. Тому, з метою отримання ще вищої пропускної здатності, пропонується реалізація даного програмного комплексу з використанням статично-компільованої мови програмування, що використовує альтернативні способи управління

пам’яттю, наприклад C з повністю ручним керуванням [35], C++ з механізмом розумних вказівників [36] або Rust з механізмом Borrow Checker [37]. Будь-яка з наведених мов програмування компілюється одразу в машинний код, що виключає накладні витрати JIT-компіляції на етапі виконання.

4.7. Висновки до розділу 4

У даному розділі було проведено комплексний порівняльний аналіз модифікованого методу контенто-залежної фрагментації даних Twin CDC та вже відомих обраних методів на чотирьох тестових наборах даних, що відображають як реальні сценарії застосування процесу дедублікації, так і наближений до найгіршого випадок з високою ентропією вмісту. Експерименти виконувалися на спеціально налаштованому хості без застосування віртуалізації, з використанням платформи .NET 9 та фіксованих параметрів усіх методів, що забезпечило відтворюваність результатів та коректність порівняння.

Аналіз метрик дедублікації показав, що модифікований метод Twin CDC забезпечує вищий коефіцієнт дедублікації даних *D* на 2-3% на двох із трьох реальних наборів даних у порівнянні з базовим Fast CDC та іншими методами, при цьому зберігаючи значення коефіцієнта девіації *V* та геометричного коефіцієнта якості фрагментації *Q* на рівні з іншими методами фрагментації. Для високоентропійного синтетичного набору усі методи очікувано показали нульове значення коефіцієнта *D*, що підтверджує наближений до найгіршого випадок для систем дедублікації.

Оцінка метрик продуктивності продемонструвала, що модифікований метод Twin CDC дозволяє суттєво підвищити пропускну здатність контенто-залежної фрагментації даних порівняно з базовим методом Fast CDC та іншими методами. Залежно від набору даних, приріст пропускної здатності фрагментації становить від 2% до 39% (без урахування високоентропійного набору, на якому зафіксовано приріст до 249%), а

пропускна здатність дедублікації зростає від 3% до 46%. Таким чином, модифікований метод Twin CDC прискорює етап фрагментації даних, при цьому зберігаючи конкурентну якість дедублікації даних, а в деяких випадках покращуючи її.

Узагальнюючи результати порівняльного аналізу, можна стверджувати, що модифікований метод Twin CDC забезпечує ефективніший баланс між якістю фрагментації (за метриками *D*, *V* та *Q*) та продуктивністю у порівнянні з розглянутими методами контенто-залежної фрагментації даних. Отримані результати дозволяють розглядати Twin CDC як перспективний кандидат для використання у виробничих системах дедублікації даних, де одночасно важливими є висока ефективність усунення дубльованих фрагментів і максимальна пропускна здатність конвеєра обробки даних. Перспективні напрями подальшої роботи охоплюють паралелізацію двонаправленого пошуку меж, оптимізацію операцій ІО та реалізацію методу статично-компільованими мовами програмування для додаткового зменшення накладних витрат часу виконання.

5. БІЗНЕС-МОДЕЛЬ СТАРТАП-ПРОЄКТУ

5.1. Опис проблеми

У сучасних інформаційних системах обсяги даних зростають експоненційно за рахунок резервного копіювання, журналювання транзакцій, зберігання медіаконтенту, артефактів збірок, контейнерних образів та іншої службової інформації. Для стримування зростання витрат на дисковий простір і мережеву інфраструктуру все ширше застосовуються технології дедублікації даних, ключовим елементом яких є контенто-залежна фрагментація. Саме етап фрагментації визначає межі фрагментів, які далі піддаються хешуванню та порівнянню, і таким чином безпосередньо впливає як на коефіцієнт дедублікації, так і на пропускну здатність.

Наявні промислові рішення здебільшого або використовують фіксовану довжину блоків, або ґрунтуються на класичних реалізаціях CDC. Такі підходи мають низку обмежень. По-перше, алгоритми контенто-залежної фрагментації часто є обчислювально дорогими: вони створюють значне навантаження на центральний процесор і оперативну пам’ять, обмежуючи продуктивність конвеєра резервного копіювання чи синхронізації даних. Це обмежує пропускну здатність системи та збільшує час резервного копіювання, що є критичним для сервісів реального часу та хмарних платформ.

По-друге, стандартні CDC-методи не забезпечують достатньо гнучкого контролю над розподілом розмірів фрагментів. На практиці це призводить до надмірної варіативності довжин блоків, тобто появи аномально малих або великих фрагментів даних, що знижує ефективність дедублікації й ускладнює планування ресурсів сховища. Для хмарних постачальників і корпоративних дата-центрів це безпосередньо означає збільшення витрат на дисковий простір, мережевий трафік та енергоспоживання при масштабуванні сервісу.

Таким чином, постає необхідність розробки модифікованого методу контенто-залежної фрагментації та програмного комплексу на його основі, які б забезпечували більш раціональний розподіл розмірів фрагментів, підвищений коефіцієнт дедублікації та вищу пропускну здатність при роботі з великими обсягами даних. Отже, програмний продукт *ChunkIt* може бути спрямований на зменшення витрат замовників на інфраструктуру зберігання, резервного копіювання і мережевої підсистеми без погіршення якості користувацького сервісу.

5.2. Визначення зацікавлених сторін

Вирішення проблем, пов'язаних з неефективною контенто-залежною фрагментацією, і, як наслідок, недостатньою ефективністю дедублікації даних у сховищах великого обсягу, є актуальним для низки ключових учасників ринку. Йдеться як про постачальників інфраструктури зберігання даних, так і про розробників програмних продуктів та кінцевих корпоративних споживачів, для яких критичними є вартість зберігання, пропускна здатність каналів та надійність резервного копіювання. Визначимо та опишемо основні зацікавлені сторони продукту *ChunkIt*, що базується на модифікованому методі Twin CDC.

Провайдери хмарних сховищ та сервісів резервного копіювання (BaaS). Постачальники хмарних рішень зацікавлені у зниженні питомих витрат на зберігання одиниці даних та мережевий трафік при реплікації й резервному копіюванні. Для них важливо мати високопродуктивний, масштабований та легко інтегрований механізм контенто-залежної фрагментації, який забезпечує вищий коефіцієнт дедублікації та пропускну здатність. Впровадження Twin CDC дозволить підвищити щільність зберігання даних та покращити SLA за рахунок скорочення вікон обслуговування та резервного копіювання.

Великі підприємства та корпорації з власними дата-центрами. Корпоративні замовники з істотними обсягами даних, такі як банківський

сектор, телеком-оператори, e-commerce, медіаплатформи, зацікавлені у зменшенні сукупної вартості володіння інфраструктурою зберігання та системами резервного копіювання. Їхня мета – скоротити обсяги збережених та переданих даних за рахунок ефективнішої дедублікації, при цьому не жертвуючи швидкодією та надійністю.

Розробники систем зберігання даних та резервного копіювання. Незалежні виробники програмного забезпечення, які створюють backup-системи, об'єктні сховища, реєстри артефактів, системи архівації чи сховища логів, зацікавлені у готовому високопродуктивному CDC-рішенню, незалежно від використаної мови програмування, системи контролю версій, CI/CD-системи тощо. Вони прагнуть вбудувати у свої продукти сучасний метод фрагментації, що підвищує якість дедублікації, не ускладнюючи архітектуру та не збільшуючи витрати на підтримку. Наявність добре задокументованого API, еталонних реалізацій та тестових наборів є критичною для цієї групи.

DevOps-інженери та SRE-фахівці. Ця категорія фахівців відповідає за проєктування та експлуатацію інфраструктури зберігання та резервного копіювання. Їх цікавить прогнозована продуктивність, контрольоване використання процесорного часу й оперативної пам'яті, а також можливість тонкого налаштування параметрів фрагментації, включаючи середній розмір фрагмента, межі відхилень та профіль навантаження на CPU/диск. Вони зацікавлені у рішенні, за яким легко спостерігати, масштабувати та інтегрувати в існуючі CI/CD-конвеєри.

Бізнес-керівники та фінансові менеджери замовників. Керівництво підприємств, фінансові директори та керівники проєктів зацікавлені насамперед у економічному ефекті від впровадження технології, а саме зменшенні витрат на дискові масиви та хмарні ресурси, скороченні витрат на мережеву інфраструктуру та підвищенні надійності резервного копіювання. Для них важливі зрозуміла модель ліцензування, прогнозований період окупності інвестицій та наявність підтверджених

експериментальних результатів, що демонструють переваги запропонованого методу над традиційними методами CDC.

Ці зацікавлені сторони формують широке коло учасників, для яких підвищення ефективності дедублікації та фрагментації даних безпосередньо пов’язане зі зниженням операційних витрат та підвищенням конкурентоспроможності IT-сервісів. Узагальнимо їх, а також відповідні інтереси та ступінь впливу у табл. 3 та 4.

Таблиця 3

Сфери зацікавленості сторін

№	Зацікавлена сторона	Сфера зацікавленості
1	Провайдери хмарних сховищ та сервісів резервного копіювання	Зменшення витрат на зберігання та трафік за рахунок вищого коефіцієнта дедублікації, підвищення пропускної здатності та надійності сервісів.
2	Великі підприємства та корпорації з власними дата-центрами	Скорочення вартості обслуговування інфраструктури зберігання та резервного копіювання, зменшення вікон резервного копіювання без втрати надійності.
3	Розробники систем зберігання даних та резервного копіювання	Інтеграція готової високопродуктивної бібліотеки CDC у власні продукти, підвищення конкурентоспроможності рішень.
4	DevOps-інженери та SRE-фахівці	Контрольоване використання ресурсів, прогнозований профіль навантаження, можливість тонкого налаштування параметрів фрагментації.
5	Бізнес-керівники та фінансові менеджери замовників	Отримання прямого економічного ефекту, зрозуміла модель ліцензування та прогнозований час окупності.

Таблиця 4

Ступені зацікавленості сторін

№	Зацікавлена сторона	В	З	Р = В × З
1	Провайдери хмарних сховищ та сервісів резервного копіювання	10	9	90
2	Великі підприємства та корпорації з власними дата-центрами	10	8	80
3	Розробники систем зберігання даних та резервного копіювання	8	8	64
4	DevOps-інженери та SRE-фахівці	6	8	48
5	Бізнес-керівники та фінансові менеджери замовників	4	6	24

Отже, найбільшу силу впливу на успіх продукту мають провайдери хмарних сховищ та сервісів резервного копіювання, а також великі підприємства й корпорації, які є основними споживачами технологій дедублікації. Саме на ці групи доцільно орієнтувати первинні пілотні впровадження, програми партнерства та маркетингову комунікацію.

Розробники систем зберігання та рішень резервного копіювання, а також системні архітектори й DevOps-фахівці формують ядро технічної аудиторії продукту. Взаємодія з ними має будуватися через технічну документацію, відкриті еталонні реалізації, участь у профільних заходах та надання доступу до SDK і тестових середовищ.

Бізнес-керівники та фінансові менеджери мають відносно менший вплив на технічні деталі, проте саме вони ухвалюють остаточні рішення щодо інвестицій у продукт. Для цієї аудиторії необхідно готувати зрозумілі економічні обґрунтування, кейси впровадження та порівняльні звіти, що демонструють зменшення витрат і підвищення ефективності завдяки використанню запропонованого продукту.

5.3. Інноваційне програмне рішення

Розроблене програмне рішення є інноваційним завдяки поєднанню модифікованого методу контенто-залежної фрагментації Twin CDC з високопродуктивною реалізацією на платформі .NET та вбудованою системою вимірювання й аналізу метрик якості дедублікації. На відміну від традиційних підходів, продукт орієнтований не лише на виконання фрагментації, а й на кероване досягнення цільового розподілу розмірів фрагментів, максимізацію коефіцієнта дедублікації та пропускної здатності під реальні робочі навантаження. Далі наведені основні характеристики даного інноваційного програмного рішення є.

Модифікований метод Twin CDC з двонапрямленим пошуком меж фрагментів. В основі рішення лежить новий модифікований метод Twin CDC, який використовує двонаправлений пошук меж фрагментів, окремі Gear-таблиці для визначення значень хешів вікон пошуку, а також встановлення альтернативних меж фрагментів. Такий підхід дозволяє зменшити дрейф границь, уникати надмірно коротких та надмірно довгих фрагментів і формувати розподіл розмірів, більш наближений до логнормального. У результаті зростає коефіцієнт дедублікації та покращується стабільність якості фрагментації на різних типах та структурах даних.

Високопродуктивна реалізація мовою C# з опорою на низькорівневі можливості платформи .NET. Алгоритмічне ядро реалізоване з використанням низькорівневих засобів керування пам’яттю, потокового читання файлів, та оптимізації розгортання циклів. Це дає змогу обробляти великі потоки даних у режимі реального часу з мінімальними накладними витратами на кероване оточення виконання і GC. Програмне рішення демонструє вищу пропускну здатність у порівнянні з класичними реалізаціями CDC, зберігаючи при цьому сумісність із типовою .NET-інфраструктурою.

Гнучка конфігуруваність профілів фрагментації під конкретні сценарії використання. Рішення підтримує налаштування параметрів фрагментації, що дає змогу оптимізувати фрагментацію під різні класи даних: резервні копії БД, журнали подій, двійкові артефакти або мультимедіа.

Орієнтація на економічний ефект та зниження вартості підтримки сховищ. Завдяки кращому коефіцієнту дедублікації та високій пропускній здатності рішення дає змогу одночасно скорочувати обсяг збережених і переданих даних та зменшувати витрати процесорного часу на фрагментацію. Це безпосередньо впливає на зниження сукупної вартості володіння сховищем і мережевою інфраструктурою, що є критично важливим для хмарних провайдерів та великих корпоративних дата-центрів.

Таким чином, інноваційність даного програмного рішення полягає у поєднанні нового модифікованого методу контенто-залежної фрагментації Twin CDC, його оптимізованої реалізації на платформі .NET та розвинених інструментів вимірювання й аналізу ефективності. Це робить продукт придатним як для вбудовування в існуючі системи зберігання та резервного копіювання, так і для створення на його основі окремого комерційного сервісу оптимізації роботи та зберігання великих обсягів даних.

5.4. Комерційний програмний продукт

Розроблене програмне рішення на основі модифікованого методу Twin CDC може бути комерційним програмним продуктом у сфері оптимізації зберігання та передачі даних для хмарних провайдерів, систем резервного копіювання, реєстрів артефактів та корпоративних сховищ. Поєднання модифікованого методу контенто-залежної фрагментації, високопродуктивної .NET-реалізації та вбудованої системи аналітики створює набір переваг, які дозволяють продукту впевнено конкурувати з класичними підходами до дедублікації.

Високий коефіцієнт дедублікації. Завдяки використанню методу Twin CDC комерційний продукт забезпечує вищий коефіцієнт дедублікації у порівнянні з традиційними методами контенто-залежної фрагментації. Вищий ступінь виявлення повторних фрагментів дозволяє істотно скоротити обсяг даних, що зберігаються та передаються, зменшуючи потребу в дисковому просторі й мережевому трафіку. Для замовника це безпосередньо означає зниження витрат на інфраструктуру зберігання та резервного копіювання.

Висока пропускна здатність та ефективне використання апаратних ресурсів. Алгоритмічне ядро, реалізоване мовою C# з використанням низькорівневих можливостей платформи .NET, забезпечує високу швидкість фрагментації та дедублікації при помірному навантаженні на процесор та оперативну пам’ять. Продукт здатний працювати поблизу граничної пропускної здатності дискової підсистеми, не обмежуючи продуктивність конвеєра резервного копіювання чи передачі даних. Це є критично важливим для хмарних провайдерів та великих підприємств, де час простою й вікно резервного копіювання безпосередньо впливають на якість сервісу.

Керований розподіл розмірів фрагментів та стабільна якість фрагментації. Особливістю модифікованого методу Twin CDC є двонаправлений пошук меж фрагментів і механізм встановлення альтернативних меж фрагментів, що дозволяє уникати аномально малих і великих фрагментів. Завдяки цьому розподіл розмірів фрагментів наближений до цільового, що покращує прогнозованість навантаження на сховище, кеші та індекси. У практичних умовах це спрощує планування ємності, налаштування політик кешування та підвищує стабільність коефіцієнта дедублікації для різних типів даних.

Гнучка інтеграція та кросплатформенність. Комерційний продукт може постачатися у вигляді вбудовуваної .NET-бібліотеки, консольної утиліти або контейнеризованого застосунку для інтеграції з існуючими

системами резервного копіювання, сховищами логів, реєстрами артефактів і CI/CD-конвеєрами. Підтримка кросплатформенності та простих сценаріїв розгортання за допомогою засобів контейнеризації знижує бар'єр входу для різних категорій замовників і дозволяє впроваджувати рішення без радикальної перебудови інфраструктури.

Вбудоване середовище тестування та апробації. Запропонований продукт включає підсистему апробації та візуалізації основних метрик продуктивності та дедублікації, включаючи коефіцієнт дедублікації, коефіцієнт девіації фрагментів, геометричний коефіцієнт якості фрагментації, а також метрики продуктивності фрагментації та дедублікації. Наявність зручних порівняльних графіків дозволяє замовнику якісно обрати набір параметрів фрагментації під конкретний випадок використання, а також підібрати найбільш економічно вигідну конфігурацію під типове навантаження системи.

Таким чином, комерційний програмний продукт на основі модифікованого методу Twin CDC вирізняється на ринку поєднанням високої ефективності зберігання, продуктивності, керованої якості фрагментації, гнучкості інтеграції та низьких витрат супроводу. Сукупність цих конкурентних переваг робить його привабливим рішенням для організацій, що прагнуть зменшити витрати на інфраструктуру зберігання та резервного копіювання, зберігаючи або підвищуючи рівень надійності та швидкодії IT-сервісів.

5.5. Сегментація ринку

Запропоновані програмні засоби націлені переважно на B2B сегмент ринку. У даному сегменті B2B можна виокремити компанії, що розробляють системи резервного копіювання, великі медіа-платформи, а також постачальників хмарного постійного сховища. Використовуючи запропоноване програмне рішення, наведені компанії матимуть можливість використати новітнє продуктивне рішення з можливістю його тонкого

налаштування саме під конкретні вимоги кожної з компаній, незалежно від їх доменної специфіки. Даний продукт легко інтегрувати у вже існуючу інфраструктуру завдяки декільком шляхам постачання, що має скоротити час на впровадження запропонованого програмного рішення, а отже і отримати переваги економії дискового простору та процесорного часу з мінімальним часовим проміжком з початку впровадження до повноцінного використання.

B2C сегмент є абсолютно вторинним відносно сегменту B2B, оскільки запропоноване програмне рішення має теоретичний та практичний поріг входження, а також вимагає наявності специфічного випадку використання. Тим не менш, кінцеві користувачі сегменту B2C зможуть відчути ефективність впровадження даного програмного продукту завдяки поліпшенню якості сервісу основного сегменту клієнтів B2B.

5.6. Унікальна ціннісна пропозиція

Унікальна ціннісна пропозиція даного продукту полягає в поєднанні науково обґрунтованого алгоритмічного вдосконалення контенто-залежної фрагментації з практично орієнтованою реалізацією для екосистеми .NET та чітко вимірюваним економічним ефектом для замовника. На відміну від більшості рішень, у яких механізми CDC є прихованим внутрішнім компонентом комплексних систем резервного копіювання, запропонований продукт позиціонується як відкритий, вбудовуваний модуль фрагментації з прозорими метриками якості дедублікації та продуктивності. Це дозволяє інтегрувати Twin CDC безпосередньо в існуючі сховища та конвеєри обробки даних, отримуючи виграш у коефіцієнті дедублікації й пропускній здатності без необхідності заміни всієї інфраструктури.

Ключовим елементом ціннісної пропозиції є керований розподіл розмірів фрагментів та стабільність параметрів фрагментації на різних класах даних. Використовуючи двонаправлений пошук меж та механізм альтернативного встановлення межі фрагментів, Twin CDC формує

фрагменти з розподілом, наближеним до цільового, що підвищує ефективність дедублікації та спрощує планування ємності сховищ. У поєднанні з високопродуктивною реалізацією мовою C# та низьким накладним навантаженням на процесор і пам’ять це забезпечує реальне зменшення сукупної вартості володіння інфраструктурою зберігання за рахунок одночасного скорочення обсягів даних і часу, необхідного на їх фрагментацію та зберігання.

Додаткову унікальність рішенню надає вбудована підсистема тестування та адаптації, що дає змогу кількісно підтверджувати заявлені переваги, оскільки замовник отримує не лише метод дедублікації, а й інструмент дослідження власних даних із доступом до розширених метрик дедублікації та продуктивності. У поєднанні з гнучкими моделями постачання це формує цілісну ціннісну пропозицію, згідно якої потенційні користувачі отримають науково обґрунтований та емпірично протестований метод, готовий до промислового використання, з прозорою економією ресурсів і мінімальним бар’єром інтеграції у вже наявні елементи інфраструктури.

5.7. Структура прибутків та витрат

Дана бізнес-модель передбачає поєднання декількох взаємодоповнювальних джерел доходу та відносно стабільну структуру операційних витрат. Це дає змогу адаптувати продукт як для великих корпоративних замовників, так і для невеликих команд розробників, що інтегрують механізми контенто-залежної фрагментації та дедублікації у власні рішення.

5.7.1. Джерела доходу

Ліцензування серверного програмного забезпечення (on-premises). Основним джерелом прибутку є продаж ліцензій на використання програмного продукту *ChunkIt* у корпоративній інфраструктурі замовників. Ліцензія може надаватись за кількістю вузлів (серверів) або обсягом

оброблюваних даних. Такий підхід є привабливим для великих підприємств і організацій з жорсткими вимогами до контролю над даними та інфраструктурою.

Підписка на службу оптимізації зберігання даних. Для клієнтів, які віддають перевагу хмарній моделі споживання, передбачено сервісну модель оплати. Користувачі мають сплачувати щомісячну або щорічну підписку залежно від фактичного обсягу даних, що фрагментуються та дедублікуються, або за кількістю інтегрованих джерел даних, наприклад, сховищ резервних копій, реєстрів артефактів тощо. Це забезпечує прогнозований регулярний грошовий потік для стартапу.

OEM-ліцензування та інтеграція з рішеннями партнерів. Частина доходів формується за рахунок OEM-моделі, за якої запропонований програмний продукт інтегрується в продукти сторонніх розробників. У цьому випадку партнер сплачує роялті або фіксовану плату за включення функціональності фрагментації та дедублікації до свого продукту. Такий канал дозволяє масштабувати ринок збуту без суттєвого збільшення власних витрат на маркетинг і продажі.

Послуги з впровадження, консалтингу та конфігурації. Окремий напрямок доходів становлять професійні послуги з аудиту існуючої інфраструктури зберігання та резервного копіювання, розробка рекомендацій щодо інтеграції продукту *ChunkIt*, налаштування профілів фрагментації під конкретні типи даних, розробка індивідуальних модулів для CI/CD-пайплайнів чи внутрішніх систем замовника. Такий формат є актуальним для великих корпоративних клієнтів з нетиповими вимогами.

Супровід, технічна підтримка та навчальні послуги. Додаткові доходи забезпечуються контрактами на розширену технічну підтримку з гарантіями часу відповіді та подовженого терміну оновлень, а також проведенням тренінгів для DevOps-команд, системних архітекторів та розробників замовника. Це має сприяти лояльності клієнтів та довгостроковій співпраці.

5.7.2. Прогнозовані витрати

Витрати на розробку та науково-дослідні роботи. До цієї категорії належать витрати на оплату праці розробників, дослідників та тестувальників, які працюють над подальшим вдосконаленням алгоритму Twin CDC та інших нових інноваційних методів фрагментації, оптимізацією реалізації програмного рішення на платформі .NET та портування іншими мовами програмування, розширенням підтримуваного набору метрик та візуалізацій, а також забезпеченням сумісності з новими версіями інструментів і платформ. Значна частка цих витрат є постійною та критично важливою для підтримки технологічної переваги продукту.

Інфраструктурні витрати. Сюди входить оренда та обслуговування хмарних ресурсів, а саме плата за віртуальні машини, сховища та мережеві сервіси, необхідних для хостингу SaaS-рішення, тестових стендів та демо-середовищ. Також враховуються витрати на системи моніторингу та логування, репозиторії артефактів, засоби безперервної інтеграції та доставки, а також резервне копіювання внутрішніх даних підприємства.

Маркетингові витрати. До цієї групи відносяться витрати на просування продукту, включаючи участь у профільних конференціях, реклама та PR-активності, підтримка сайту продукту, підготовка демонстраційних матеріалів, участь у програмах для стартапів та партнерських ініціативах. Окремим елементом є витрати на роботу відділу продажів і бізнес-розвитку, включаючи підготовку комерційних пропозицій та супровід переговорів з корпоративними клієнтами.

Операційні витрати та підтримка користувачів. До операційних витрат належать витрати на технічну підтримку, регулярні оновлення програмного забезпечення, виправлення помилок та забезпечення сумісності з новими версіями операційних систем, платформ віртуалізації та хмарних сервісів. У цю ж категорію можна віднести витрати на внутрішню документацію, керування релізами та забезпечення інформаційної безпеки.

Адміністративні та юридичні витрати. До цієї категорії належать витрати на адміністративний, бухгалтерський і юридичний персонал, реєстрацію інтелектуальної власності (патенти, торгові марки тощо), потенційні ліцензійні платежі за використання сторонніх компонентів, а також витрати, пов’язані з дотриманням нормативних вимог у різних юрисдикціях.

Податкове навантаження. До цієї категорії належать обов’язкові податкові платежі, пов’язані з діяльністю стартап-проєкту, включаючи податок на прибуток підприємств, податки і збори з фонду оплати праці, а також непрямі податки, що виникають при наданні послуг клієнтам залежно від юрисдикції (податок на додану вартість тощо).

5.8. Модель ощадливого стартапу

З метою узагальнення всіх вищенаведених елементів бізнес-моделі продукту *ChunkIt*, сформуємо її у вигляді шаблонного опису бізнес-моделі ощадливого стартапу, що фокусується на проблемах клієнтів, рішеннях та гіпотезах. Дана модель дозволить швидко випробувати бізнес-гіпотезу про успішність запропонованого проєкту, базуючись на основних факторах, що можуть вплинути на його успіх:

- цільова аудиторія, споживачі та їх основні категорії;
- найбільш поширені проблеми цільової аудиторії;
- рішення, що вирішує наведені проблеми цільової аудиторії;
- унікальна ціннісна пропозиція, що вирізняє продукт на ринку;
- прихована перевага, яку конкурентам буде важко скопіювати;
- ключові метрики відстежування;
- канали розповсюдження та популяризації продукту;
- потенційні потоки доходу;
- структура необхідних витрат.

У табл. 5 наведено сформовану модель ощадливого стартапу, що включає в себе всі наведені фактори впливу.

Таблиця 5

Модель ощадливого стартапу *ChunkIt*

<table>
<tr><td rowspan="2">Проблема
Низький ступінь дедублікації, великий час фрагментації даних, високе споживання ресурсів під час фрагментації, постійна потреба у розширенні постійного сховища даних.</td><td>Рішення
Модифікований метод Twin CDC та його програмна реалізація, що забезпечує вищий коефіцієнт дедублікації, а також вищу пропускну здатність фрагментації та дедублікації даних.</td><td rowspan="2">Унікальна ціннісна пропозиція
Готове рішення контенто-залежної фрагментації даних, що легко інтегрувати у вже існуючу інфраструктуру.</td><td>Прихована перевага
Вбудоване середовище тестування та апробації дозволить виконувати тонке налаштування фрагментації під конкретний випадок використання.</td><td rowspan="2">Споживачі
Провайдери хмарних сховищ та сервісів резервного копіювання, великі підприємства та корпорації з власними дата-центрами, розробники систем зберігання даних та резервного копіювання, DevOps-інженери та SRE-фахівці, бізнес-керівники та фінансові менеджери замовників</td></tr>
<tr><td>Ключові метрики
Кількість проданих ліцензій, кількість поновлених ліцензій, відгуки та пропозиції користувачів.</td><td>Канали
Через відділ продажів, маркетингові кампанії (реклама), виступи та демо на конференціях з розробки ПЗ.</td></tr>
<tr><td colspan="2">Структура витрат
Утримання штату (заробітна плата, бонуси), податкове навантаження, подальші розробка та дослідження, маркетингові кампанії та промоутинг, хмарна інфраструктура.</td><td colspan="3">Потоки доходу
Продаж ліцензій різних типів та їх поновлення, підтримка споживачів, технічний супровід та аудит.</td></tr>
</table>

5.9. Висновки до розділу 5

У даному розділі було сформульовано та структуровано бізнес-модель стартап-проєкту *ChunkIt*, що ґрунтується на модифікованому методі контенто-залежної фрагментації Twin CDC та його програмній реалізації. На основі аналізу проблеми неефективної фрагментації й дедублікації даних

у сучасних системах зберігання обґрунтовано доцільність створення нового спеціалізованого програмного рішення, орієнтованого на підвищення коефіцієнта дедублікації, пропускної здатності та керованості розподілом розмірів фрагментів. Було визначено коло ключових зацікавлених сторін та проаналізовано їхні інтереси й ступінь впливу на впровадження запропонованого продукту.

Також, було показано, що розроблене інноваційне програмне рішення поєднує алгоритмічні переваги методу Twin CDC з високопродуктивною реалізацією на платформі .NET та вбудованою системою збору й візуалізації метрик якості фрагментації і дедублікації. На тлі відомих аналогів апаратно-програмних платформ дедублікації, комплексних backup-систем та відкритих інструментів було окреслено ринкову нішу для продукту, який виступає як вбудовуваний модуль контенто-залежної фрагментації даних з прозорими характеристиками й гнучкими можливостями інтеграції. Сформульовано унікальну ціннісну пропозицію, що полягає у поєднанні кращих кількісних показників, а саме коефіцієнт дедублікації, пропускна здатність та стабільність розподілу розмірів фрагментів, з безпосереднім економічним ефектом для замовників за рахунок зниження сукупної вартості володіння інфраструктурою зберігання.

Окремо було окреслено структуру прибутків та витрат, яка передбачає декілька потенційних джерел доходів та профіль витрат. Узагальнення цих елементів у формі канви бізнес-моделі показало внутрішню узгодженість технічного рішення та економічних механізмів його комерціалізації. Отже, запропонований метод Twin CDC і програмний комплекс *ChunkIt* мають не лише наукову новизну, але й потенціал до створення життєздатного комерційного продукту з перспективою масштабування на ринку систем зберігання та резервного копіювання даних.

ВИСНОВКИ

Дана магістерська дисертація присвячена дослідженню процесу дедублікації даних з основним фокусом на розробці нового методу контенто-залежної фрагментації даних шляхом модифікації одного з вже відомих методів фрагментації. В контексті даного дослідження було проаналізовано низку класичних та новітніх методів контенто-залежної фрагментації даних, охарактеризовано принципи їхньої роботи та виділено їхні переваги та недоліки. За результатами проведеного аналізу було виявлено, що класичні методи фрагментації даних демонструють конкурентний рівень дедублікації даних, проте мають помірний рівень продуктивності та мають характерний недолік утворення аномально малих та великих фрагментів даних. Новітні ж методи фрагментації мають вищу продуктивність, однак вимагають тонкого налаштування параметрів, а при фрагментації монотонних чи випадкових даних продуктивність та якість дедублікації цих методів різко знижується.

На основі проведеного аналізу було запропоновано модифікований метод контенто-залежної фрагментації даних Twin CDC, що базується на методі Fast CDC та має три ключові особливості, а саме двонаправлений пошук меж фрагментів, використання двох окремих Gear-таблиць та встановлення альтернативних меж фрагментів. Завдяки двонаправленому пошуку меж, а саме перевазі кандидатним позиціям, що знаходяться в околі цільового середнього розміру фрагментів, досягається утворення більшої кількості фрагментів, розмір яких наближений до цільового. Підвищення ймовірності встановлення меж на монотонних даних досягається шляхом декореляції вікон пошуку меж з використанням двох окремих Gear-таблиць. Характерний базовому методу недолік утворення великої кількості аномально великих фрагментів даних було усунуто завдяки механізму встановлення альтернативних меж фрагментів.

З метою підтвердження переваги запропонованого методу над вже відомими, було розроблено програмний комплекс *ChunkIt*, що реалізує

розглянуті та модифікований методи контенто-залежної фрагментації даних, а також повний конвеєр збору та візуалізації метрик продуктивності та дедублікації. Даний програмний комплекс реалізовано мовою програмування C# 13.0 на платформі .NET 9 та забезпечує неупереджене тестування методів фрагментації шляхом проведення повторюваних бенчмарків продуктивності та дедублікації. Крім того, було детально описано архітектуру створеного програмного комплексу, висвітлено структуру та призначення його модулів, а також принцип збору та розрахунку кожної з впроваджених метрик.

З використанням розробленого програмного комплексу, було виконано комбінований емпіричний порівняльний аналіз метрик продуктивності та дедублікації запропонованого методу у порівнянні з відомими методами фрагментації. З метою диверсифікації отриманих результатів було використано чотири різних тестових набори даних, що відтворюють як реальні сценарії використання дедублікації, так і наближений до найгіршого випадок з високою ентропією вмісту. Аналіз метрик дедублікації показав, що запропонований метод демонструє підвищення коефіцієнту дедублікації в межах 2-3% порівняно з іншими методами, при цьому зберігаючи конкурентні значення коефіцієнту девіації фрагментів та геометричного коефіцієнту якості фрагментації даних. Оцінка метрик продуктивності продемонструвала, що запропонований метод дозволяє істотно збільшити обсяг даних, які фрагментуються та ефективно усуваються за одиницю часу. Так, приріст пропускної здатності фрагментації даних складає від 2% до 39% в залежності від тестового набору даних, а на високоентропійному наборі даних приріст складає до 249%. В той же час, приріст пропускної здатності дедублікації складає від 3% до 46% в залежності від набору даних. Сукупний аналіз метрик продуктивності та дедублікації свідчить, що запропонований метод Twin CDC має більш збалансований компроміс між обсягом дедублікованих даних, розподілом розмірів фрагментів даних та загальною продуктивністю.

Крім того, було розроблено бізнес-модель стартап-проєкту *ChunkIt*, що ґрунтується на запропонованому методі Twin CDC та його програмній реалізації. На основі аналізу проблеми помірної ефективності фрагментації та дедублікації у сучасних системах зберігання було обґрунтовано доцільність створення інноваційного програмного рішення, орієнтованого на підвищення коефіцієнта дедублікації, пропускної здатності та керованості розподілом розмірів фрагментів. Визначено ключових зацікавлених сторін, їхні інтереси та ступінь впливу, сформовано унікальну ціннісну пропозицію, що полягає в поєднанні кращих кількісних показників з прямим економічним впливом у вигляді зниження сукупної вартості володіння інфраструктурою зберігання даних. Також було окреслено структуру доходів і витрат та зведено модель ощадливого стартапу.

Підсумовуючи, можна зробити висновок, що поставлену мету даного дослідження було досягнуто та всі його основні завдання було виконано. Підтверджена ефективність запропонованого методу контенто-залежної фрагментації даних робить його приданим до впровадження у реальних виробничих середовищах, зокрема у системи резервного копіювання, медіа-платформи та в інфраструктуру постачальників хмарного сховища.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. How much data is generated every day? [Електронний ресурс] // SOAX Research – Режим доступу до ресурсу: https://soax.com/research/data-generated-per-day.
2. YouTube Press [Електронний ресурс] // YouTube – Режим доступу до ресурсу: https://blog.youtube/press.
3. Dynamic Prime Chunking Algorithm for Data Deduplication in Cloud Storage // KSII Transactions on Internet and Information Systems. — 2021. — Vol. 15, No. 4. — P. 1342–1359.
4. Convergent Encryption Enabled Secure Data Deduplication Algorithm for Cloud Environment. Concurrency and Computation: Practice and Experience / Ahmad Sh., Arif M., Ahmad J., Mehfuz S. — 2024.
5. A Study of Practical Deduplication / Meyer D. T., Bolosky W. J. // 9th USENIX Conference on File and Storage Technologies. — 2011. — P. 1–13.
6. VectorCDC: Accelerating Data Deduplication with Vector Instructions / Sreeharsha Udayashankar, Abdelrahman Baba, Samer Al-Kiswany // Proceedings of the 23rd USENIX Conference on File and Storage Technologies. — 2025.
7. A Mathematical Theory of Communication / Shannon C. E. // Bell System Technical Journal. — 1948. — Vol. 27. — P. 379–423, 623–656.
8. SeqCDC: Hashless Content-Defined Chunking for Data Deduplication // Sreeharsha Udayashankar, Abdelrahman Baba, Samer Al-Kiswany — 2024.
9. Data Deduplication System Based on Content-Defined Chunking Using Bytes Pair Frequency Occurrence / Saeed A.S.M., George L.E. // Symmetry. — 2020. — Vol. 12, No. 11.
10. Bimodal Content Defined Chunking for Backup Streams / Kruus E., Ungureanu C., Dubnicki C. // Proceedings of the 8th USENIX Conference on File and Storage Technologies. — 2010. — P. 239–252.

11. An Information-Theoretic Analysis of Deduplication / Urs Niesen // IEEE Transactions on Information Theory — 2017.
12. A Thorough Investigation of Content-Defined Chunking Algorithms for Data Deduplication / Marcel Gregoriadis, Leonhard Balduf, Björn Scheuermann, Johan Pouwelse — 2024.
13. Data Fingerprinting with Similarity Digests / Breitinger F., Baier H. // Advances in Digital Forensics IX. — IFIP Advances in Information and Communication Technology. — 2013. — Vol. 410 — P. 3–21.
14. Handbook of Applied Cryptography / Menezes A.J., Van Oorschot P.C., Vanstone S.A. // Boca Raton: CRC Press. — 1996. — 780 p.
15. A low-bandwidth network file system / A. Muthitacharoen, B. Chen, and D. Maziere // Proceedings of the eighteenth ACM symposium on Operating systems principles. — 2001. — Vol. 35 — P. 174–187.
16. SLIMSTORE: A Cloud-based Deduplication System for Multi-version Backups / Zhang Z., Hu H., Xue Z., Chen C., Yu Y., Fu C., Zhou X., Li F. // Proc. 37th IEEE International Conference on Data Engineering. — 2021. — P. 1841–1846.
17. The Design of Fast Content-Defined Chunking for Data Deduplication Based Storage Systems / Xia W., Zhou Y., Jiang H., Feng D., Hua Y., Hu Y., Zhang Y., Liu Q. // IEEE Transactions on Parallel and Distributed Systems. — 2020. — Vol. 31, No. 8. — P. 1889–1905.
18. The Design of Fast Content-Defined Chunking for Data Deduplication Based Storage Systems / Xia W., Hua Y., Jiang H., Feng D., Tian L., Fu M. // IEEE Transactions on Parallel and Distributed Systems. — 2020. — Vol. 31, No. 9. — P. 2020–2037.
19. AE: An Asymmetric Extremum Content Defined Chunking Algorithm for Fast and Bandwidth-Efficient Data Deduplication / Zhang Yucheng, Jiang Hong, Feng Dan, Xia Wen, Fu Min, Huang Fangting, Zhou Yukun // Proceedings of IEEE INFOCOM 2015. – IEEE, 2015. – P. 1337–1345.

20. A new content-defined chunking algorithm for data deduplication in cloud storage / Widodo R.N.S., Lim H., Atiquzzaman M. // Future Generation Computer Systems. — 2017. — Vol. 71. — P. 145–156.
21. Foundation – Introducing Content Defined Chunking (CDC) [Електронний ресурс] // Restic Blog – Режим доступу до ресурсу: https://restic.net/blog/2015-09-12/restic-foundation1-cdc.
22. FastCDC: A Fast and Efficient Content-Defined Chunking Approach for Data Deduplication / Xia Wen, Jiang Hong, Feng Dan, Hua Yujuan, Hu Yucheng, Zhou Yukun, Zhang Yong // 2016 USENIX Annual Technical Conference. — 2016. — P. 101–114.
23. Optimizing Compilers for Modern Architectures: A Quantitative Approach / Allen R., Kennedy K. // San Francisco, CA, USA: Morgan Kaufmann Publishers. — 2002. — 790 p.
24. Does the content defined chunking really solve the local boundary shift problem? / Tian W., Li R., Xu Z., Xiao W. // 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC). — 2017. — P. 1–8.
25. Array Class [Електронний ресурс] // Microsoft Learn – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/api/system.array
26. .NET 6 and Devirtualization [Електронний ресурс] // Ascendle – Режим доступу до ресурсу: https://ascendle.com/ideas/net-6-and-devirtualization/
27. BenchmarkDotNet Index [Електронний ресурс] // BenchmarkDotnet – Режим доступу: https://benchmarkdotnet.org/index.html
28. .NET Runtime: Tiered Compilation [Електронний ресурс] // Microsoft GitHub – Режим доступу: https://github.com/dotnet/runtime/blob/main/docs/design/features/tiered-compilation.md
29. Fast Splittable Pseudorandom Number Generators / Steele G.L. Jr., Lea D., Flood C.H. // OOPSLA ’14: Proceedings of the 2014 ACM International

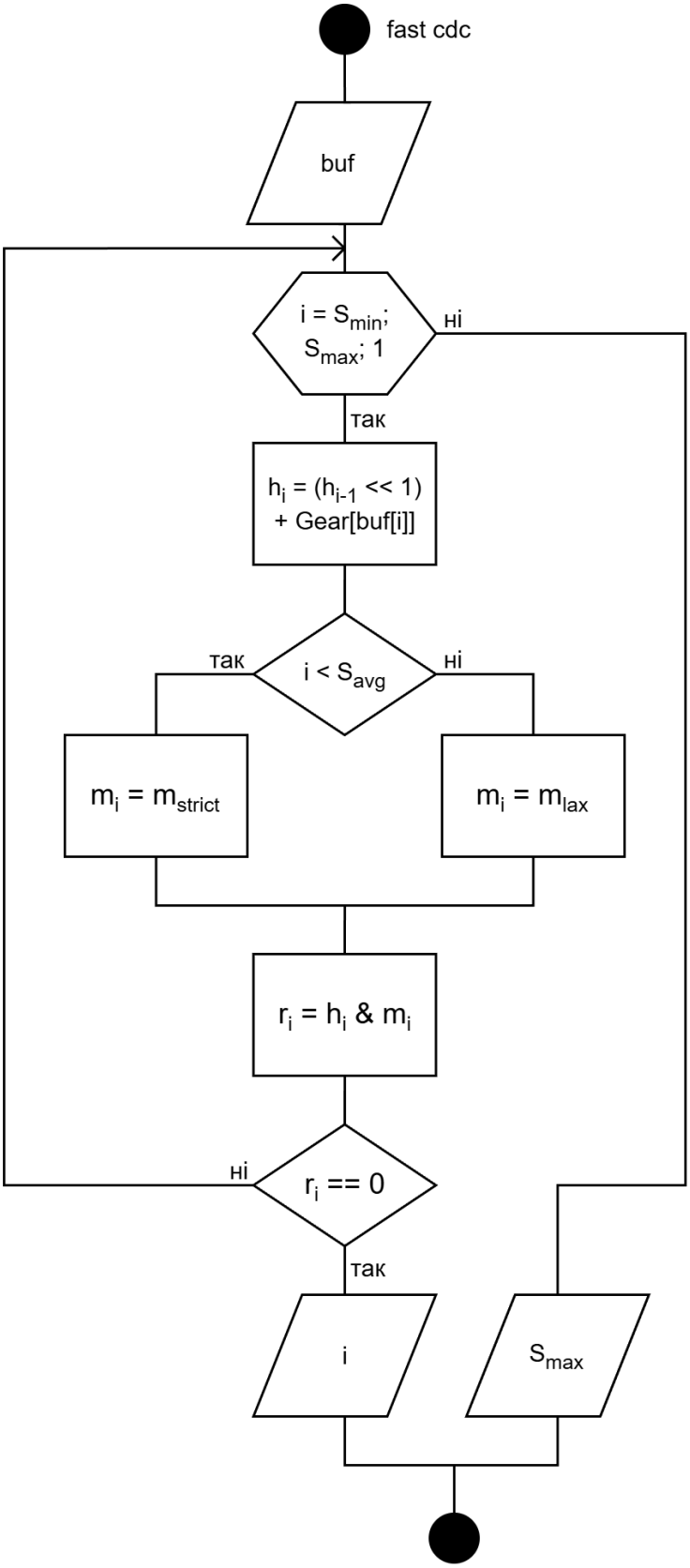
Conference on Object-Oriented Programming, Systems, Languages, and Applications. — New York: ACM, 2014. — P. 453–472.

30. Dedupbench: A benchmarking tool for data chunking techniques / Liu, A., Baba, A., Udayashankar, S., Al-Kiswany, S. // 2023 IEEE Canadian Conference on Electrical and Computer Engineering. — 2023. — P. 469–474.
31. FileOptions Enum [Електронний ресурс] // Microsoft Learn – Режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.io.fileoptions.
32. Модифікований метод та програмне забезпечення для контенто-залежної фрагментації даних / Хіцко Я.В., Калашников-Травін В.В. // Прикладна математика та комп'ютинг. ПМК, 2025: зб. тез доп. XVIII наук. конф. магістрантів та аспірантів (Київ, 19-21 лист. 2025 р.). — К.: ПТФ «Просвіта», 2025. — С. 455-459.
33. Memory-mapped files [Електронний ресурс] // Microsoft Learn – Режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/io/memory-mapped-files.
34. Managed execution process [Електронний ресурс] // Microsoft Learn – Режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/managed-execution-process.
35. Memory Management in C and Rust: A Comparative Analysis [Електронний ресурс] // Medium. — 2024. — Режим доступу: https://medium.com/@rustincode.dev/memory-management-in-c-and-rust-a-comparative-analysis-6d62807b9f0b
36. Smart Pointers [Електронний ресурс] // CppReference – Режим доступу: https://en.cppreference.com/book/intro/smart_pointers
37. References and Borrowing [Електронний ресурс] // Rust Lang – Режим доступу: https://doc.rust-lang.org/1.8.0/book/references-and-borrowing.html

ДОДАТКИ

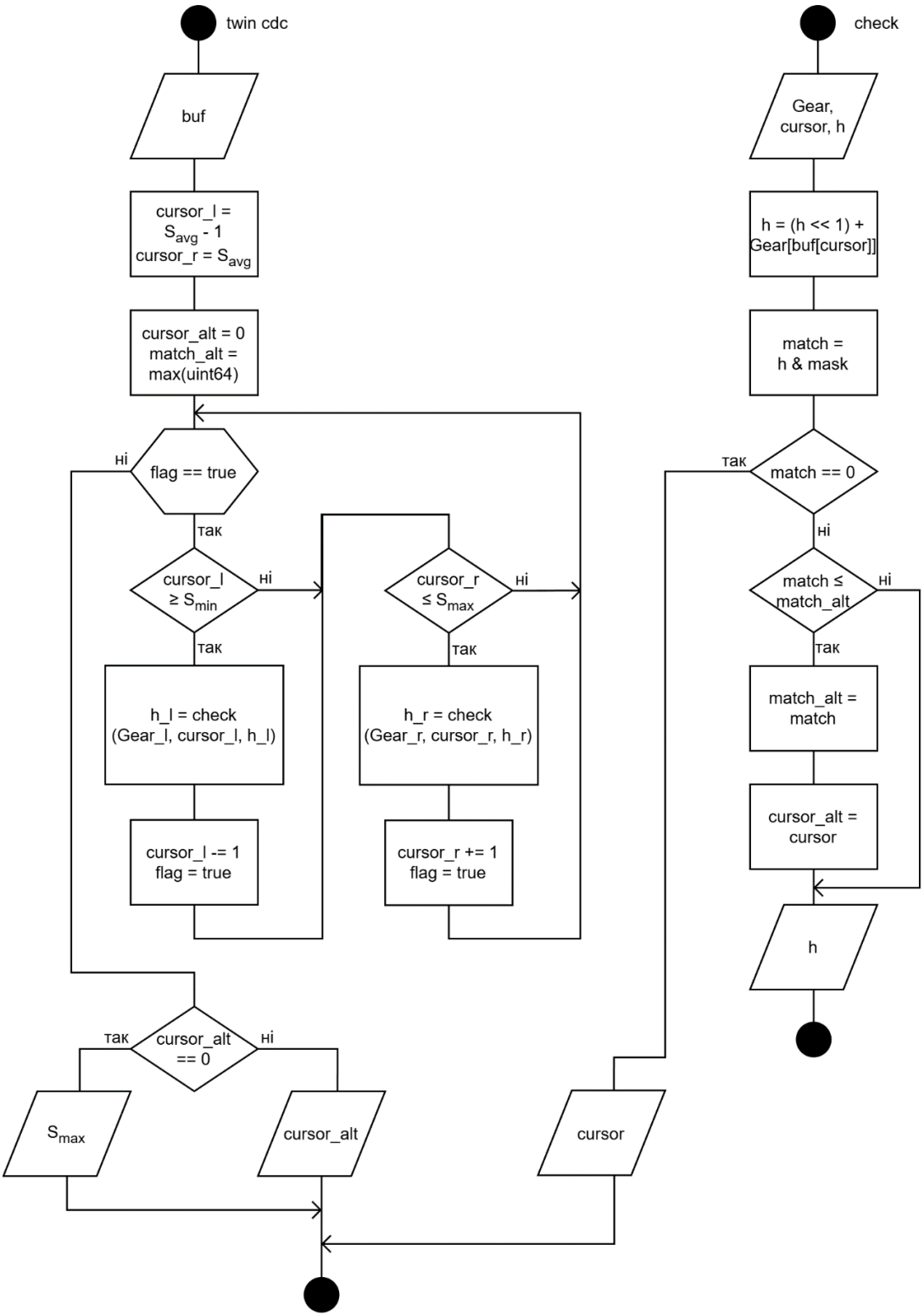
Додаток 1

Копії графічних матеріалів



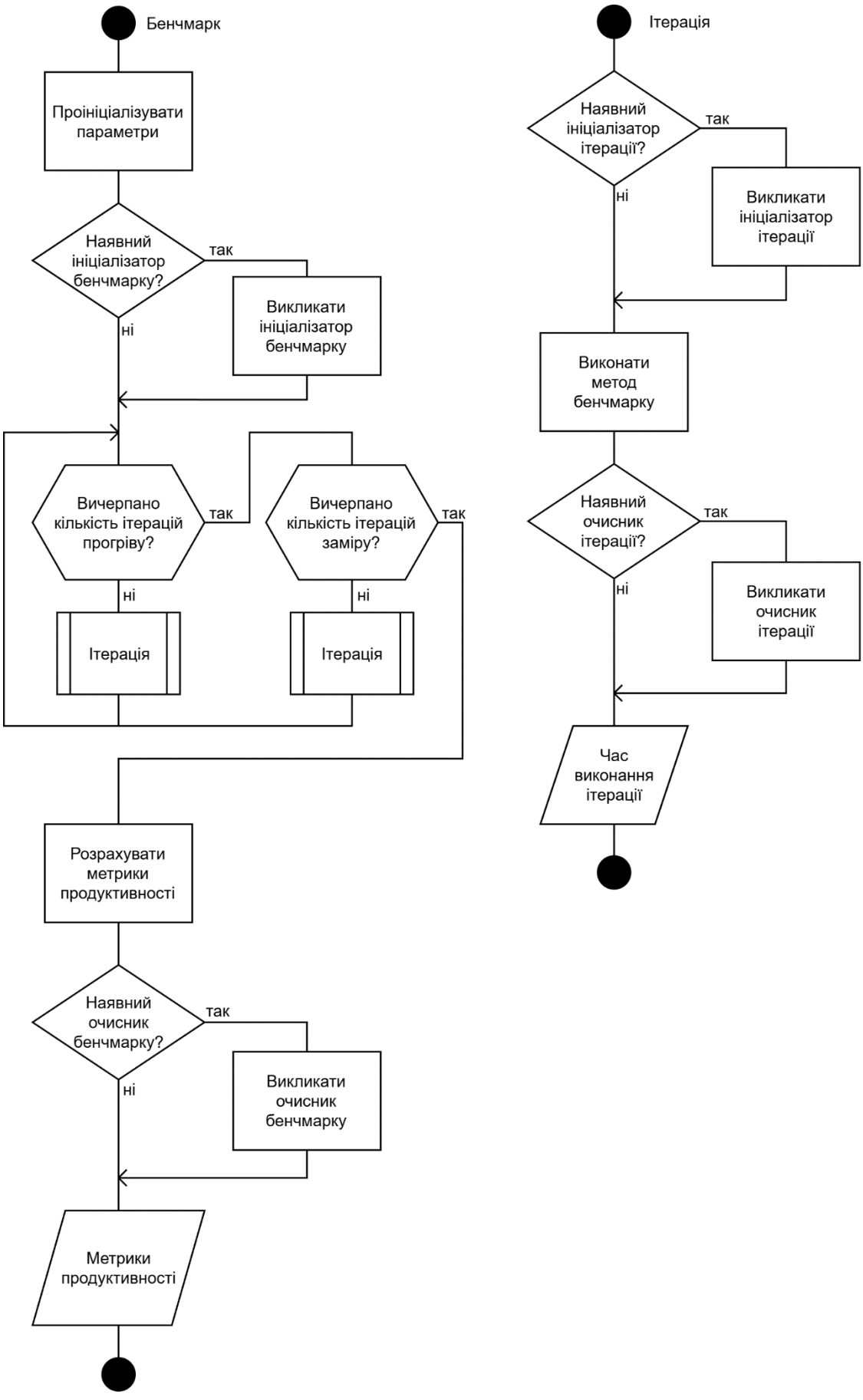
Блок-схема базового методу Fast CDC

Калашников-Травін В. В., КП-41мп



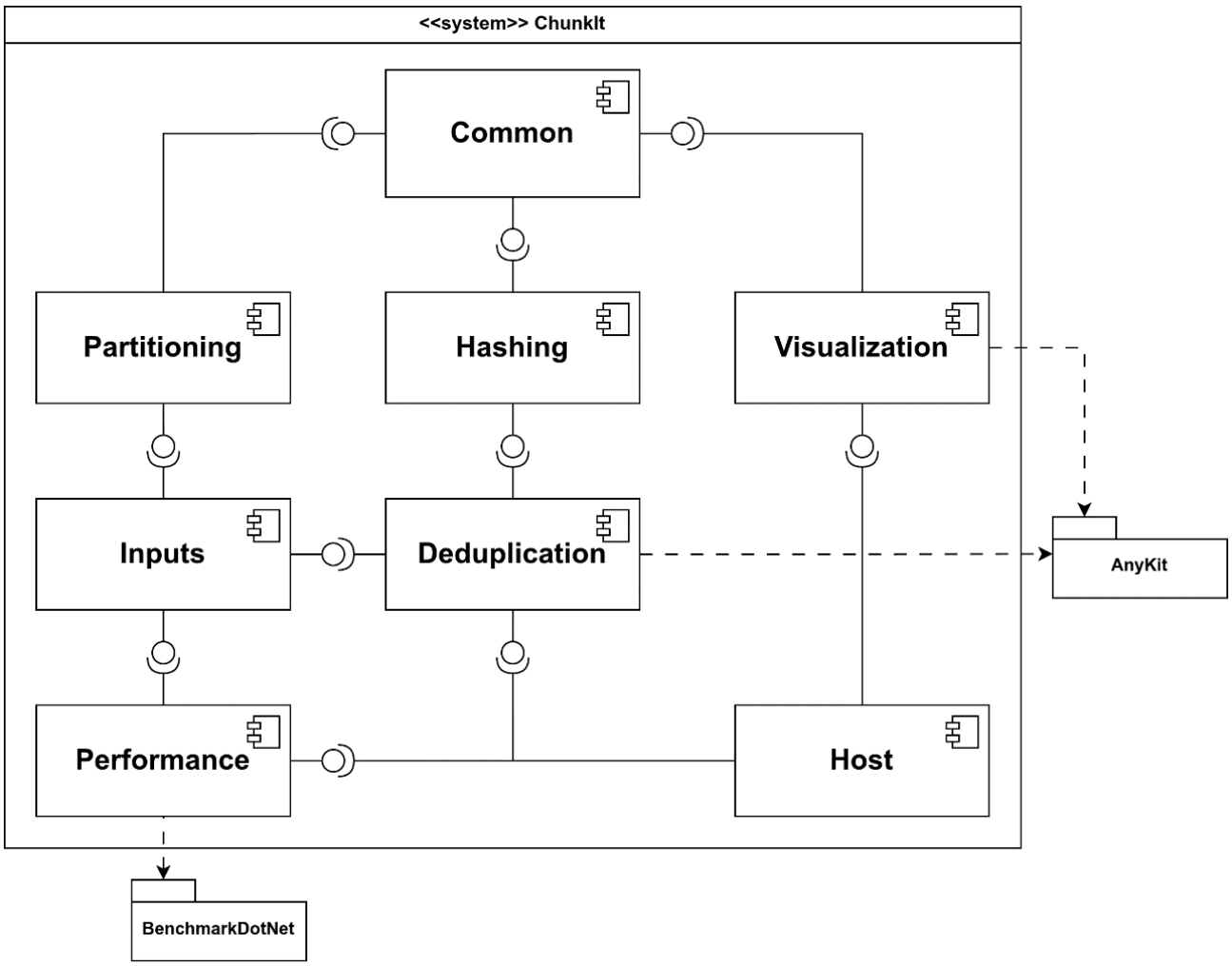
Блок-схема модифікованого методу Twin CDC

Калашников-Травін В.В., КП-41мп



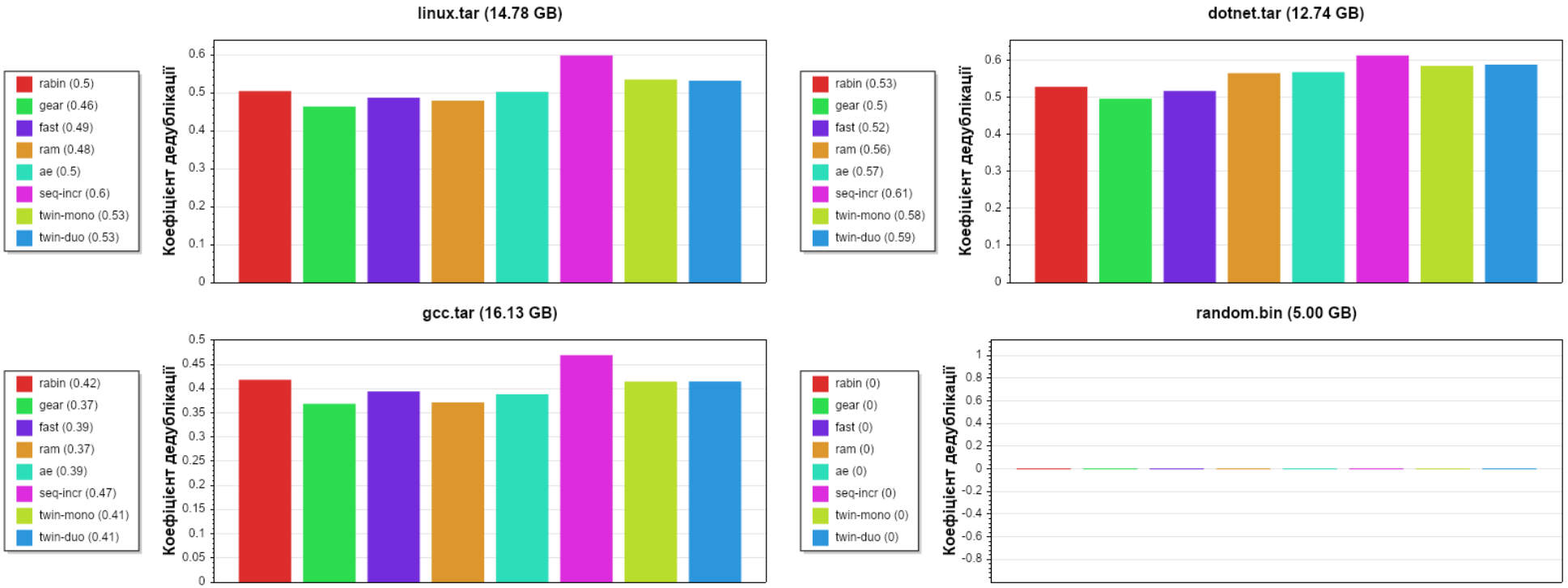
Блок-схема виконання бенчмарків продуктивності BenchmarkDotNet

Калашников-Травін В.В., КП-41мп



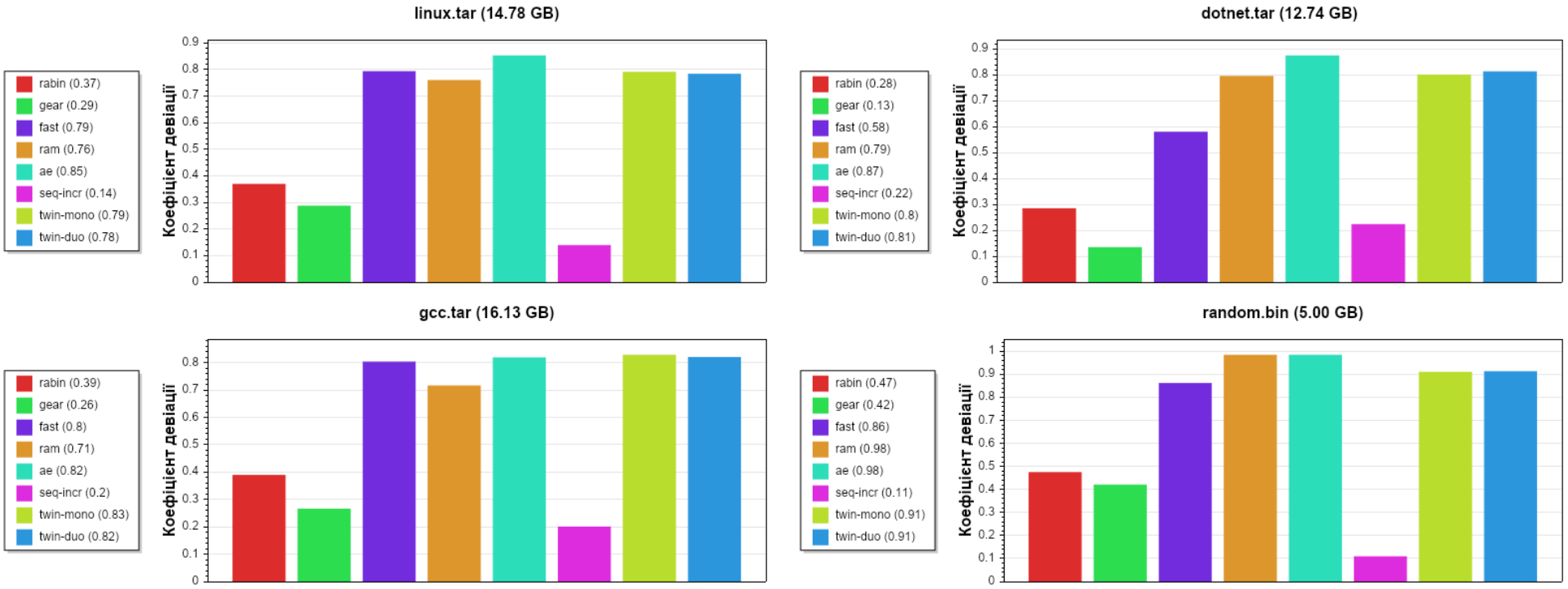
Архітектура розробленого програмного комплексу

Калашников-Травін В.В., КП-41мп



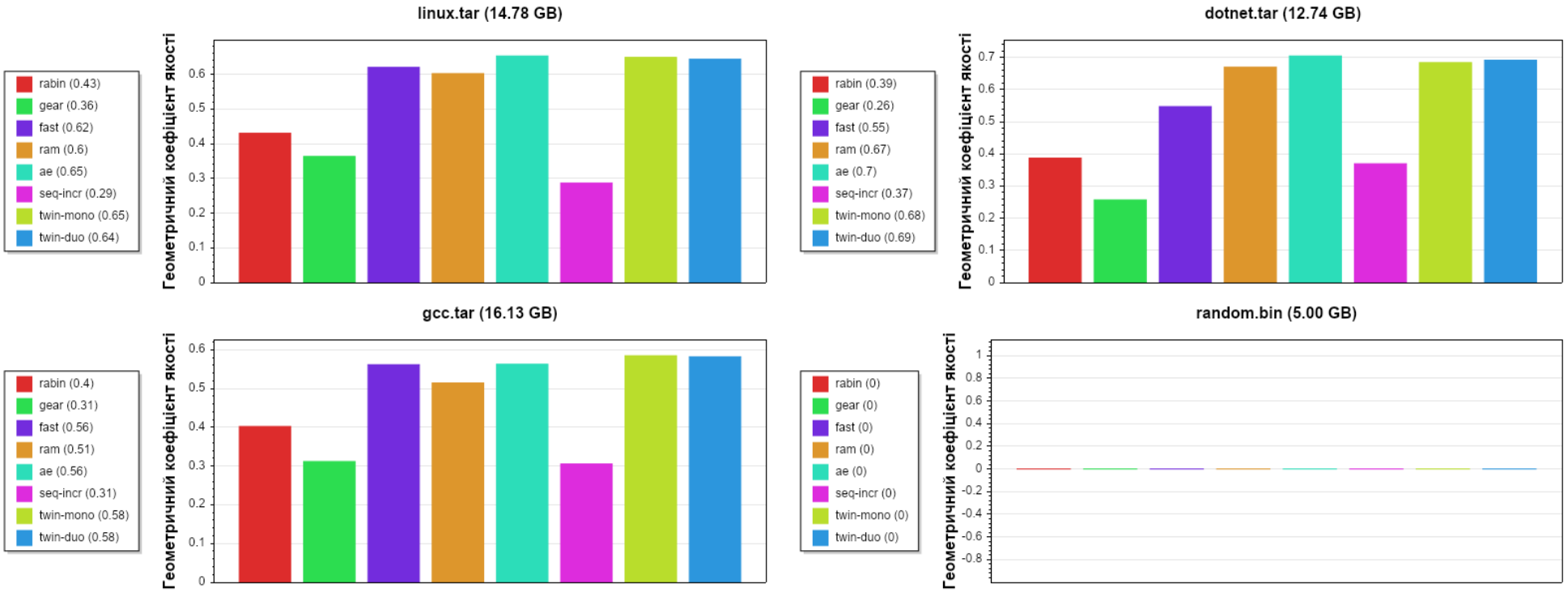
Порівняльний графік коефіцієнтів дедублікації даних

Калашников-Травін В.В., КП-41мп



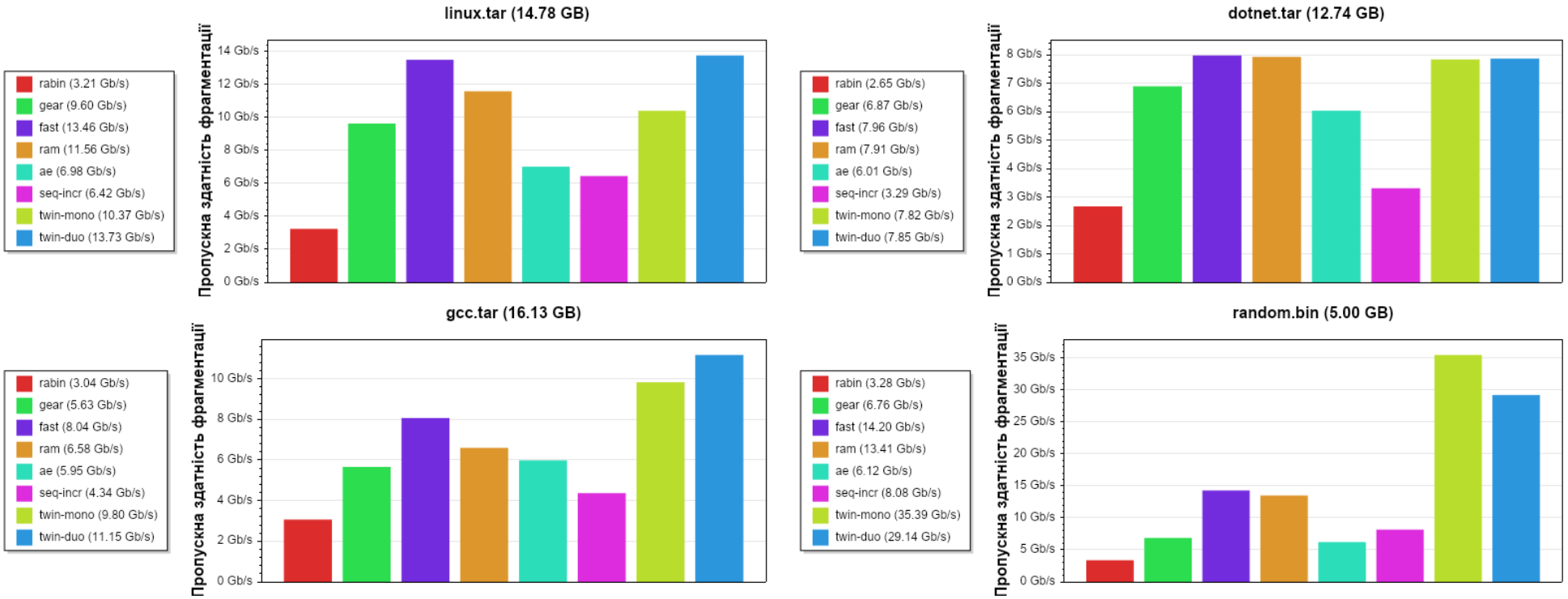
Порівняльний графік коефіцієнтів девіації фрагментів даних

Калашников-Травін В.В., КП-41мп



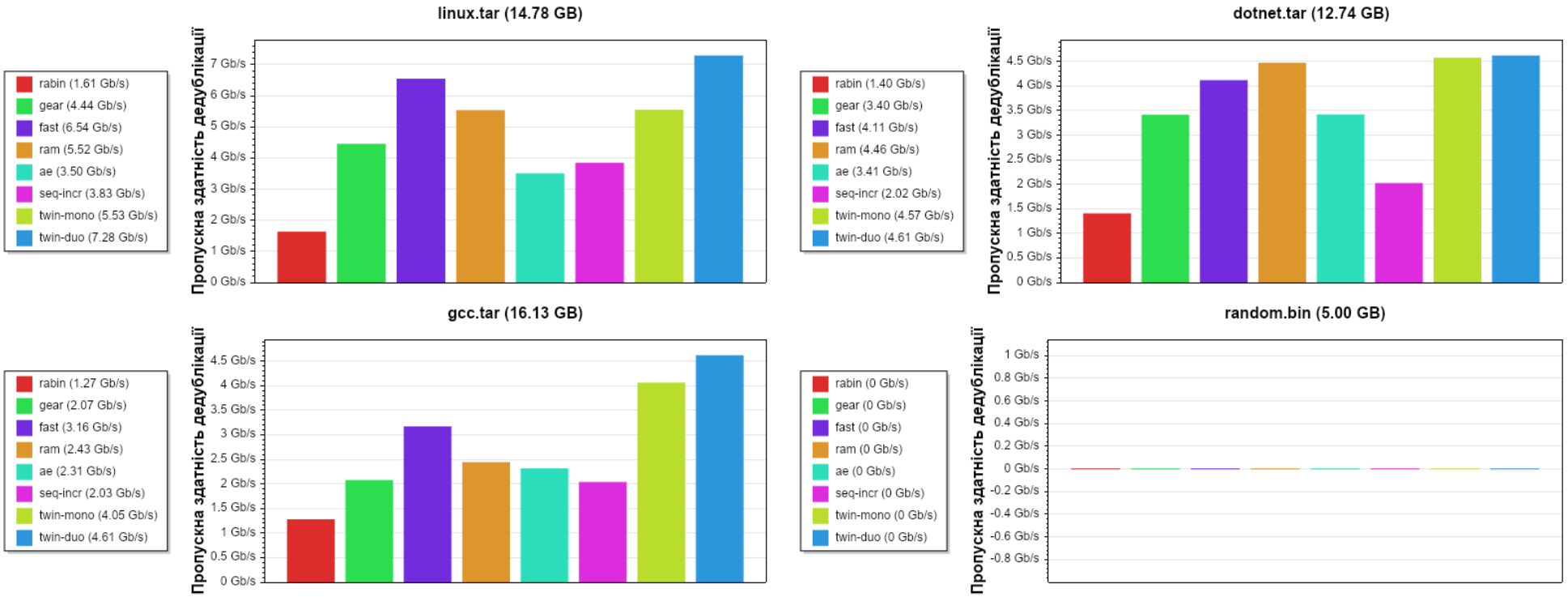
Порівняльний графік геометричних коефіцієнтів якості фрагментації даних

Калашников-Травін В.В., КП-41мп



Порівняльний графік пропускної здатності фрагментації даних

Калашников-Травін В.В., КП-41мп



Порівняльний графік пропускної здатності дедублікації даних

Калашников-Травін В.В., КП-41мп

Додаток 2

Лістинги фрагментів коду програмного комплексу

Лістинг 1. Метод Fast CDC

Вхідні дані: вихідний потік байтів

Вихідні дані: межа фрагменту

```
mid = Savg
upper = Smax
k = ceiling(log2(Savg))
strict_mask = 1 << (k + norm_level) - 1
lax_mask = 1 << (k - norm_level) - 1
cursor = Smin
fingerprint = 0
while cursor < mid do
    value = buffer[cursor]
    fingerprint = (fingerprint << 1) + Gear[value]
    match = fingerprint & strict_mask
    if match == 0 then return cursor
end while
while cursor < upper do
    value = buffer[cursor]
    fingerprint = (fingerprint << 1) + Gear[value]
    match = fingerprint & lax_mask
    if match == 0 then return cursor
end while
return upper
```

Лістинг 2. Метод Twin CDC

Вхідні дані: вихідний потік байтів

Вихідні дані: межа фрагменту

```
upper = Smax
mid = Savg
k = ceiling(log2(Savg))
mask = 1 << (k - norm_level) - 1
leftPrint = rightPrint = 0
leftCursor = mid - 1
rightCursor = mid
alternativeCursor = 0
alternativeMatch = max(uint64)
while flag do
    flag = false
    if leftCursor >= Smin then
        value = buffer[leftCursor]
        leftPrint = (leftPrint << 1) + Gearleft[value]
        match = leftPrint & mask
        if match == 0 then return leftCursor
        if match < alternativeMatch then
            alternativeMatch = match
            alternativeCursor = leftCursor
        end if
        leftCursor -= 1
        flag = true
    end if
    if rightCursor < upper then
        value = buffer[rightCursor]
        rightPrint = (rightPrint << 1) + Gearright[value]
        match = rightPrint & mask
        if match == 0 then return rightCursor
        if match < alternativeMatch then
            alternativeMatch = match
            alternativeCursor = rightCursor
        end if
        rightCursor += 1
        flag = true
    end if
end while
if alternativeCursor != 0 then
    return alternativeCursor
else
    return upper
```

Лістинг 3. Оголошення методів бенчмарків

```
public class ExampleBenchmark
{
    [Benchmark(Baseline = true)]
    public void DoWork_First() { ... }


    [Benchmark]
    public void DoWork_Second() { ... }
}
```

Лістинг 4. Параметризація бенчмарків

```
public class ExampleBenchmark
{
    [Params(1, 2, 3)]
    public int InlineParam { get; set; }


    [ParamsSource(nameof(EnumerateValues))]
    public string ExternalParam { get; set; }


    public static IEnumerable<string> EnumerateValues()
    {
        yield return “first”;
        yield return “  second ”.Trim();
        yield return “third”.ToUpper();
    }
}
```

Лістинг 5. Структура фрагменту даних

```
public readonly struct Chunk
{
    public long Id { get; }
    public long Offset { get; }
    public int Length { get; }
    public byte[] Hash { get; }
}
```

Лістинг 6. Спільний програмний інтерфейс хешування вмісту фрагментів даних

```
public interface IHasher
{
    byte[] Hash(ReadOnlySpan<byte> bytes);
}
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

МОДИФІКОВАНИЙ МЕТОД ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КОНТЕНТО-ЗАЛЕЖНОЇ ФРАГМЕНТАЦІЇ ДАНИХ

Доповідач: Калашников-Травін Владислав Володимирович

Науковий керівник: к.т.н., ст. вик. ПЗКС Хіцко Яна Володимирівна

Київ – 2025

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

- **Стрімке зростання обсягів створення та копіювання даних** у сучасному діджиталізованому середовищі призводить до збільшення витрат на зберігання та передачу інформації.
- **Дедублікація даних** є одним із ключових процесів економії постійного сховища та пропускної здатності межеревих каналів передачі даних, а її ефективність безпосередньо залежить від якості першого етапу - **контенто-залежної фрагментації даних (CDC)**.
- Наявні методи контенто-залежної фрагментації або не забезпечують достатньої пропускної здатності, або утворюють значну частку аномально малих чи надто великих фрагментів, що погіршує роботу систем дедублікації. **Отже, дослідження та вдосконалення методів CDC є актуальним та має простір для мінімізації відомих проблем фрагментації даних.**

ОБ’ЄКТ, ПРЕДМЕТ ТА МЕТОДИ ДОСЛІДЖЕННЯ

Об’єкт дослідження: Процес дедублікації даних.

Предмет дослідження: Способи та методи контенто-залежної фрагментації даних.

Методи дослідження: Теоретичний та емпіричний аналіз відомих методів контенто-залежної фрагментації даних.

ПОСТАНОВКА ЗАДАЧІ

Мета роботи: збільшення продуктивності фрагментації даних, підвищення коефіцієнту дедублікації даних, а також запобігання утворенню аномально малих та великих фрагментів даних.

Окремі завдання:

- провести аналіз відомих методів фрагментації;
- порівняти їх характеристики, виокремити переваги та недоліки;
- розробити нову модифікацію методу з поліпшеними характеристиками;
- створити програмну реалізацію запропонованого методу;
- реалізувати програмне забезпечення для емпіричного порівняння методів;
- провести порівняльний аналіз;
- розробити бізнес-модель.

ГІПОТЕЗА

Завдяки використанню двонаправленого пошуку меж фрагментів у поєднанні з двома окремими Gear-таблицями, а також встановленню альтернативних меж фрагментів можна досягти вищої пропускної здатності за умови збереження рівного або вищого коефіцієнту дедублікації.

ТЕРМІНОЛОГІЯ

Дедублікація - процес вилучення повторюваних фрагментів з вхідного потоку даних з метою збереження постійного сховища та мережевого трафіку.

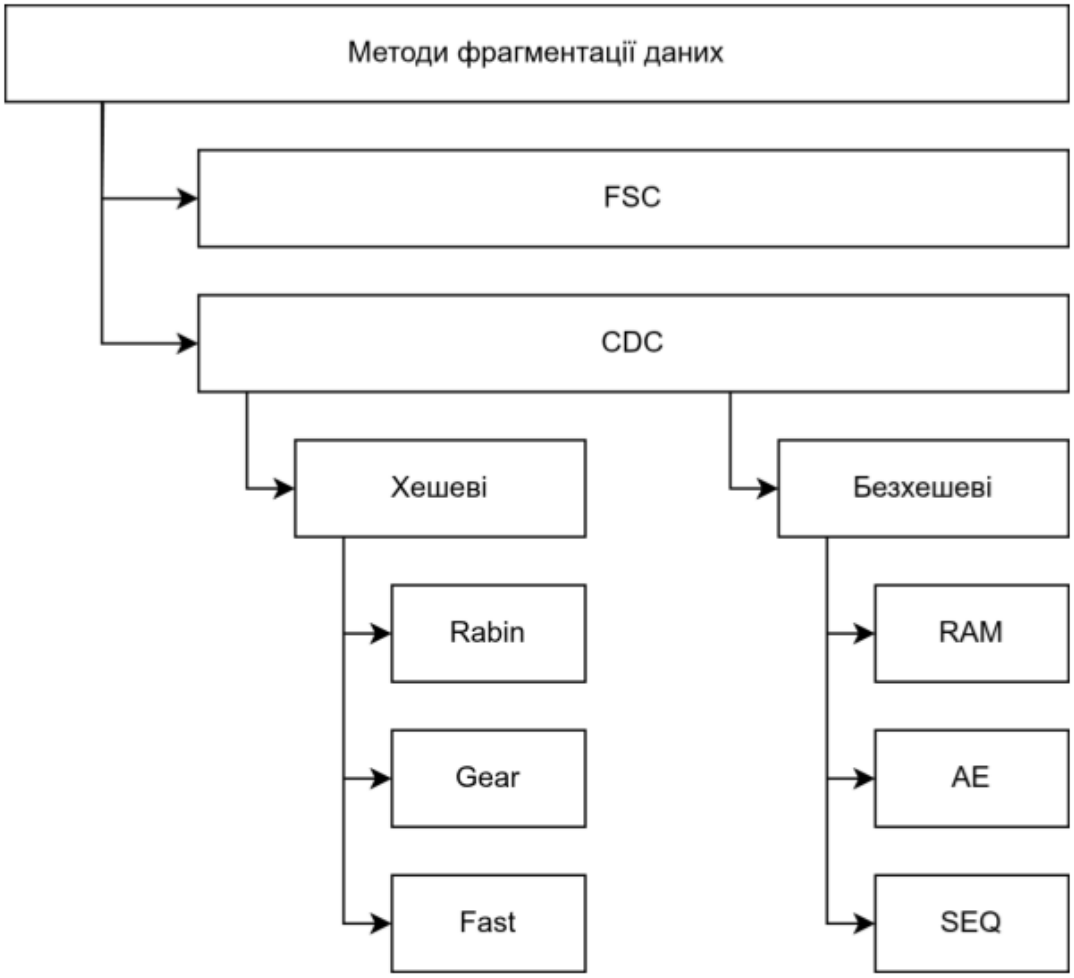
Контенто-залежна фрагментація даних (*CDC*) - перший та ключовий етап процесу дедублікації, за якого вхідний потік даних перетворюється на послідовність фрагментів даних за ознаками вмісту цього потоку.

Цільовий розмір фрагментів (S_{conf}) - один з параметрів методів контенто-залежної фрагментації, що задається значеннями мінімального (S_{min}), середнього (S_{avg}), та максимального (S_{max}) розмірів фрагментів.

Коефіцієнт дедублікації - відношення обсягу дедублікованих даних до обсягу вхідного потоку даних.

Пропускна здатність фрагментації/дедублікації даних - обсяг фрагментованих/дедублікованих даних за одиницю часу.

АНАЛІЗ ВІДОМИХ МЕТОДІВ ФРАГМЕНТАЦІЇ ДАНИХ

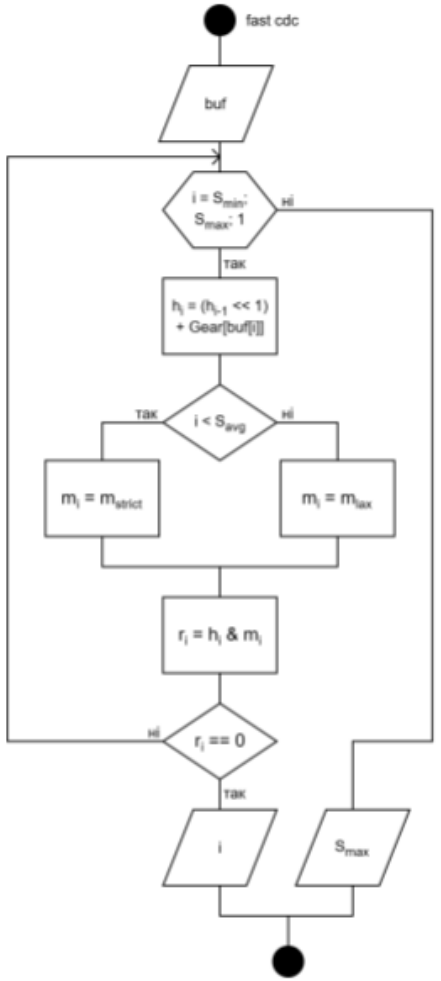


БАЗОВИЙ МЕТОД
FastCDC

FastCDC використовує таблицю Gear для розрахунку ковзного хешу вмісту даних та нормалізацію розмірів фрагментів даних.

На кожній ітерації розраховується ковзний хеш h_i, на який накладається маска m, результатом чого є залишок r_i.

Межа встановлюється на зсуві, де залишок r_i дорівнює нулю або примусово на зсуві S_{max}.

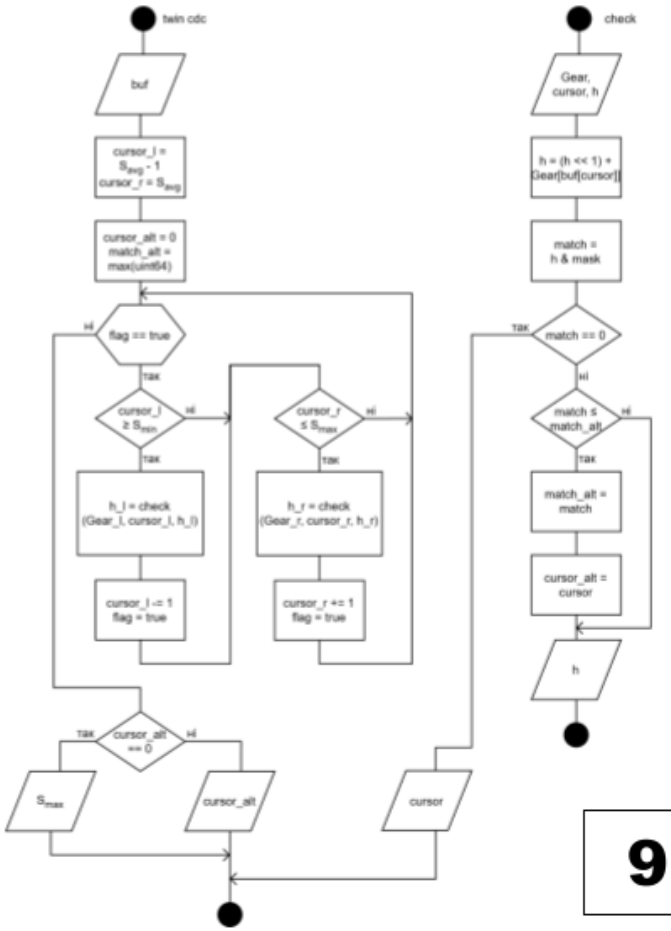


МОДИФІКОВАНИЙ МЕТОД TwinCDC [1/4]

Запропонований метод TwinCDC, що базується на методі FastCDC, покликаний збільшити пропускну здатність фрагментації та дедублікації даних, при цьому зберігши коефіцієнти дедублікації даних та девіації фрагментів даних на рівні з розглянутими методами CDC.

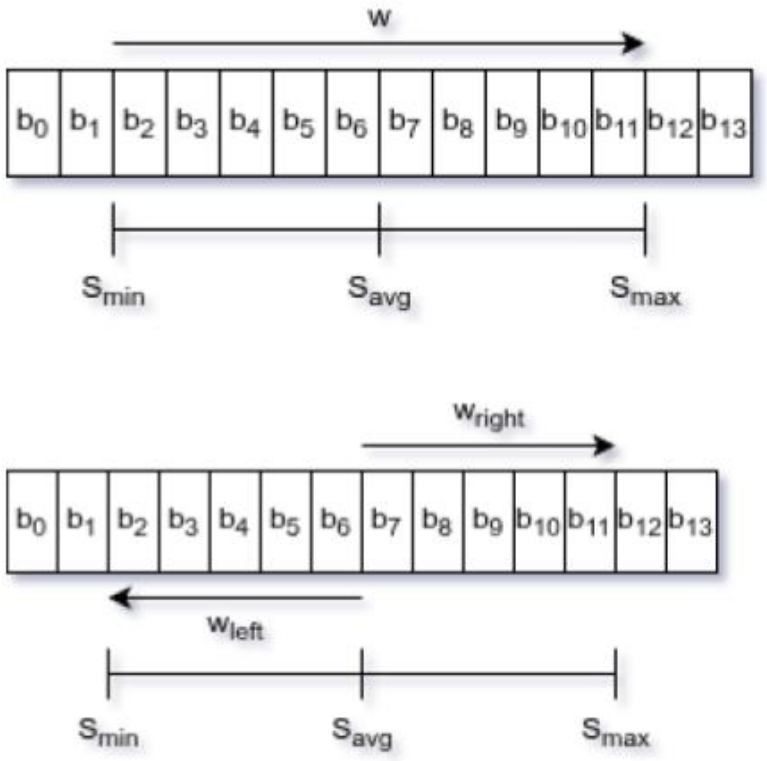
Характерними рисами TwinCDC є:

- двонаправлений пошук меж фрагментів даних
- використання окремих Gear-таблиць
- встановлення альтернативних меж фрагментів даних



МОДИФІКОВАНИЙ МЕТОД TwinCDC [2/4]

Двонаправлений пошук меж фрагментів даних пріоритетно розглядає позиції в околі середнього очікуваного розміру фрагментів S_{avg}.



МОДИФІКОВАНИЙ МЕТОД TwinCDC [3/4]

Використання окремих Gear-таблиць для лівого і правого вікон пошуку меж фрагментів дозволяє незалежно розраховувати значення ковзних хешів.

На структурованих або монотонних даних, значення ковзних хешів може бути рівним в обох вікнах пошуку, отже вікна корелюють між собою.

Декореляція вдвічі підвищує ймовірність встановлення межі ближче до S_{avg} завдяки незалежності значень ковзних хешів в обох вікнах пошуку.

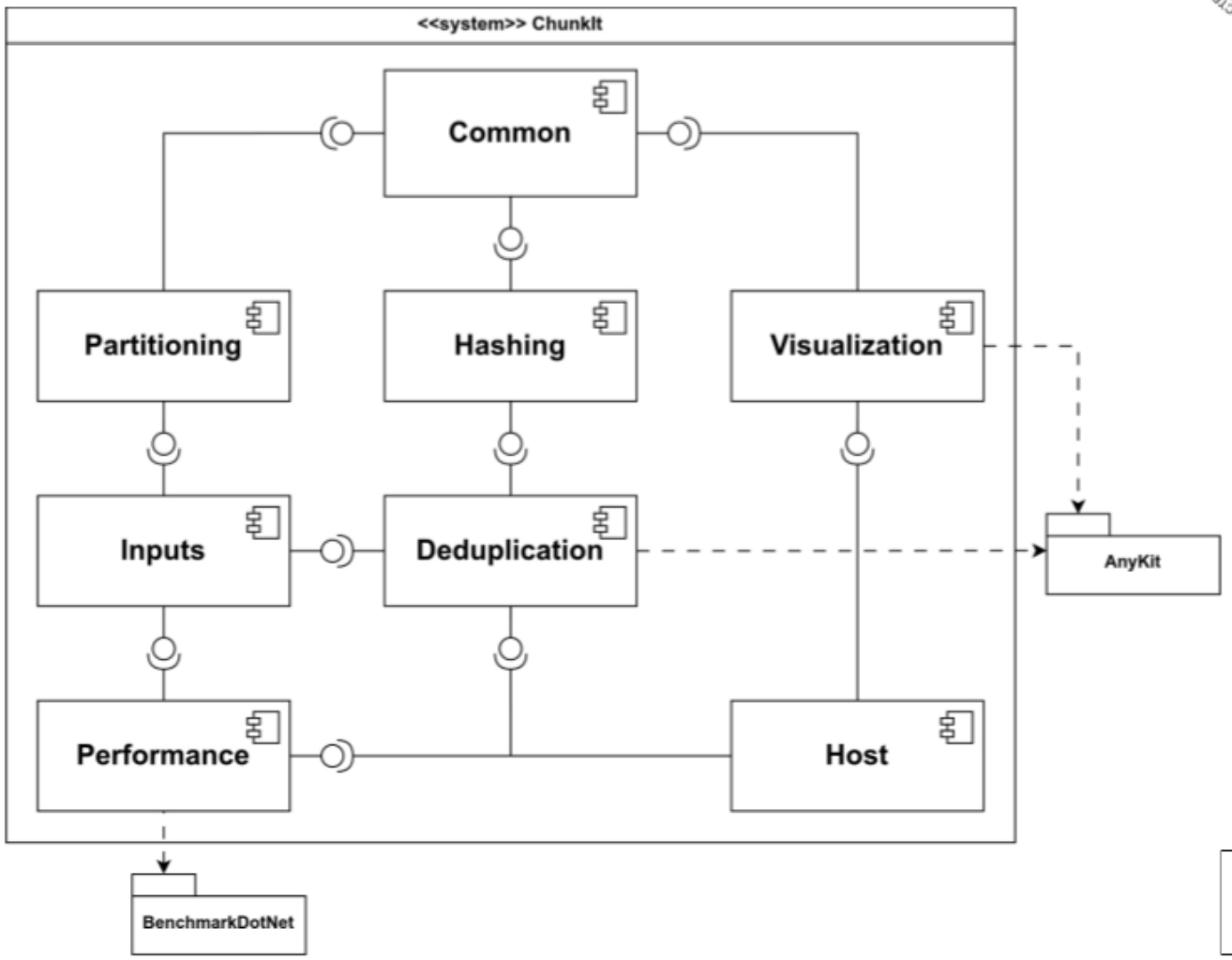
МОДИФІКОВАНИЙ МЕТОД TwinCDC [4/4]

- У відомих алгоритмах CDC, межі фрагментів даних встановлюються примусово на зсуві S_{max}, якщо предикат встановлення межі не спрацював жодного разу у заданому вікні пошуку.
- Велика кількість фрагментів розміром S_{max} негативно впливає на коефіцієнт дедублікації.

- Альтернативні межі у Twin CDC встановлюються на такому зсуві, де результат накладення маски на ковзний хеш є мінімальним.
- Такий підхід реалізує механізм примусових меж, але за зсувом ближче до S_{avg} та далі від S_{max}.

ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ЕМПІРИЧНОГО ПОРІВНЯННЯ МЕТОДІВ CDC

АРХІТЕКТУРА РОЗРОБЛЕНОГО ПРОГРАМНОГО КОМПЛЕКСУ



ПАРАМЕТРИ МЕТОДІВ CDC ТА НАБОРИ ДАНИХ ДЛЯ ПОРІВНЯЛЬНОГО АНАЛІЗУ

Набори даних:

- *linux kernel* (останні 10 версій, архів tar, **15 GB**);
- *dotnet runtime* (останні 15 версій, архів tar, **13 GB**);
- *gcc compiler platform* (останні 25 версій, архів tar, **17 GB**);
- *випадкові дані* (неструктурований бінарний потік, **5 GB**)

Хешеві алгоритми:

- rabin
- gear
- fast

Безхешеві алгоритми:

- ram
- ae
- seq-incr

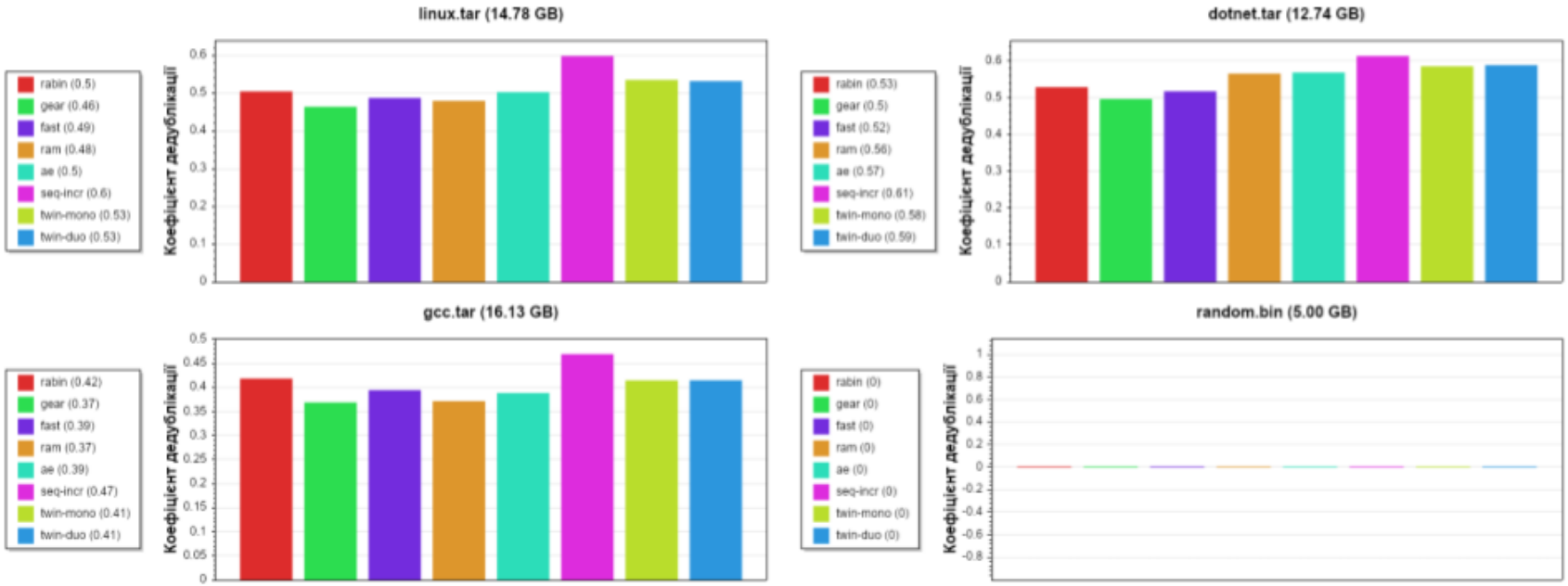
Twin CDC:

- twin-mono
- twin-duo

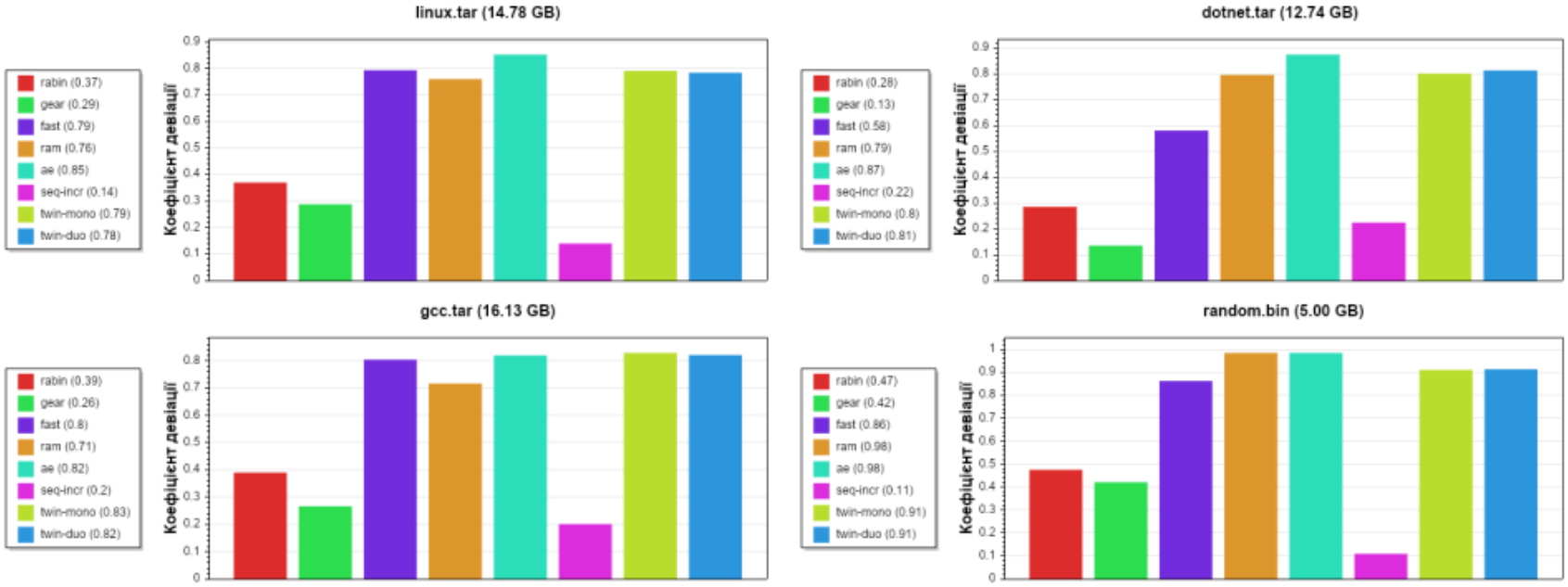
Цільовий розмір фрагментів S_{conf}:

- $S_{min} = 8$ KB
- $S_{avg} = 16$ KB
- $S_{max} = 32$ KB

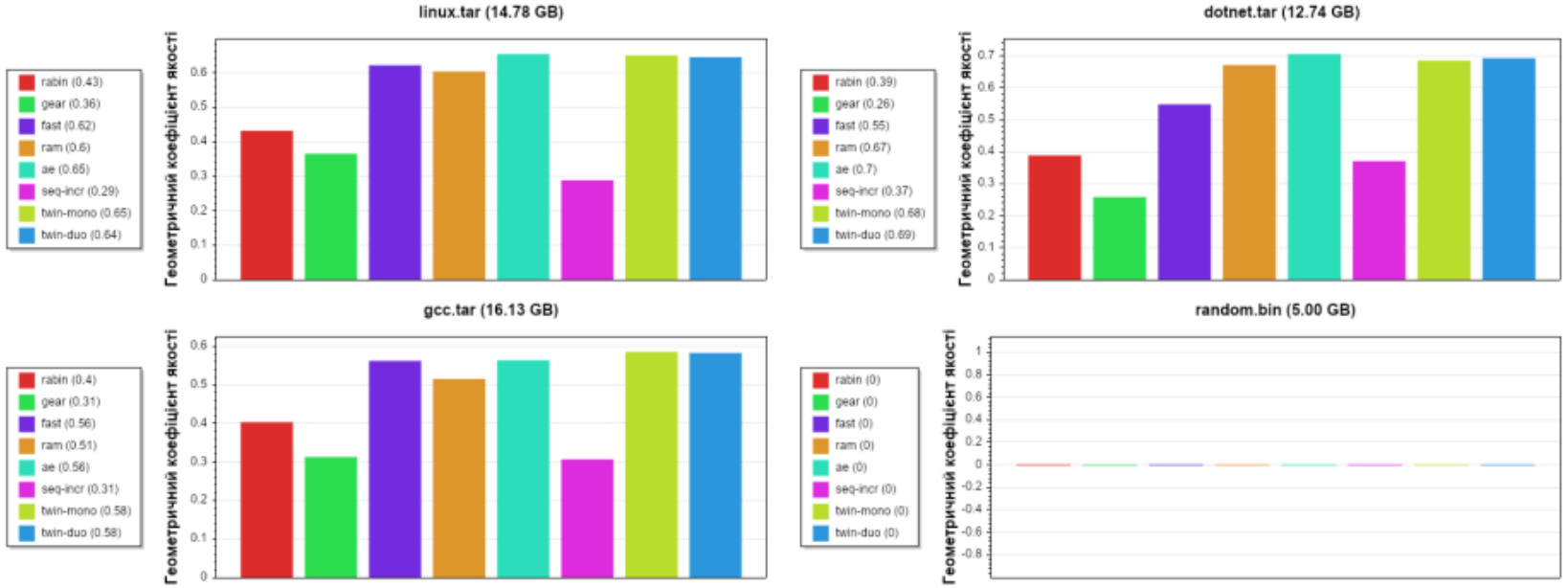
ПОРІВНЯННЯ КОЕФІЦІЄНТІВ ДЕДУБЛІКАЦІЇ ДАНИХ



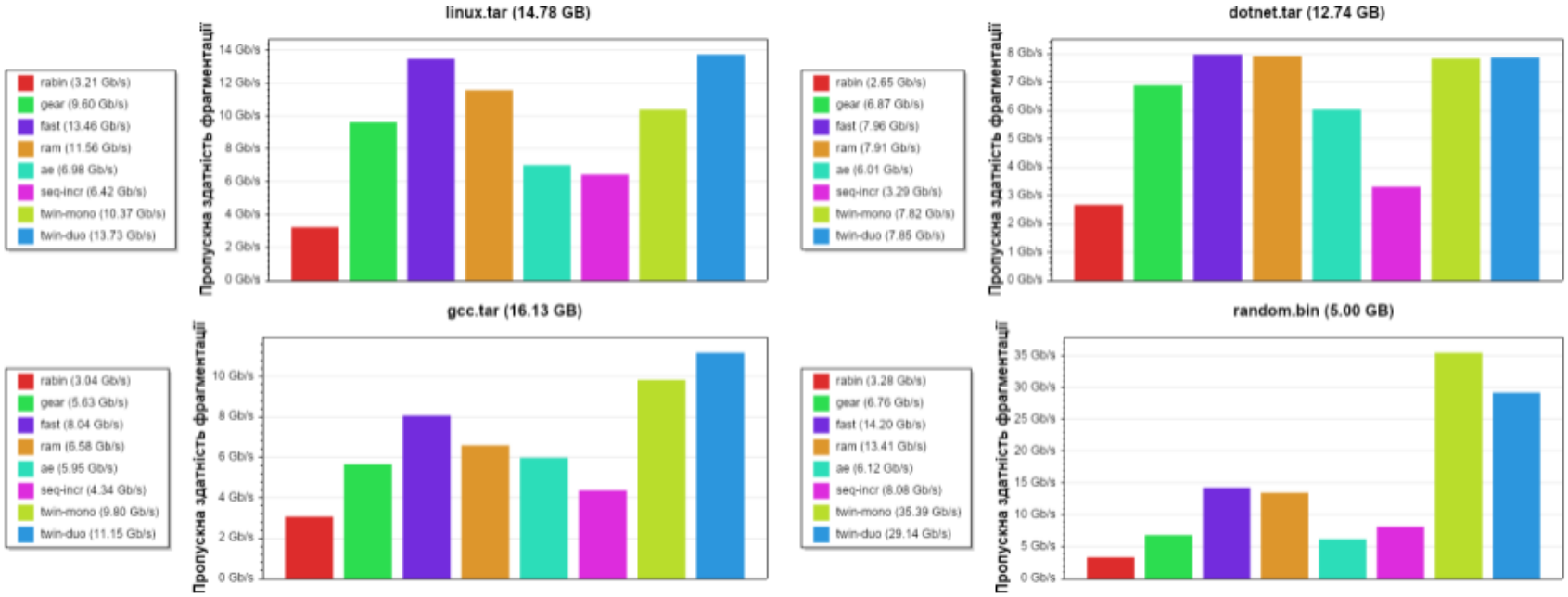
ПОРІВНЯННЯ КОЕФІЦІЄНТІВ ДЕВІАЦІЇ ФРАГМЕНТІВ ДАНИХ



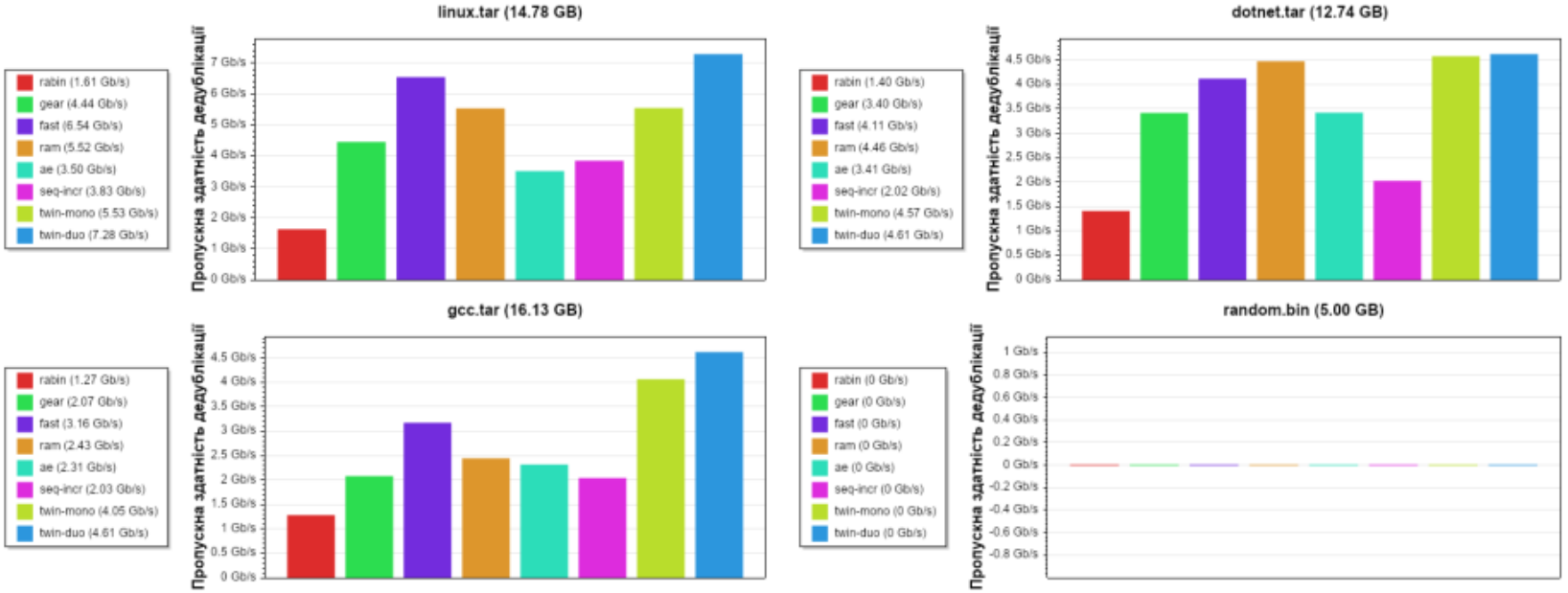
ПОРІВНЯННЯ ГЕОМЕТРИЧНИХ КОЕФІЦІЄНТІВ ЯКОСТІ ФРАГМЕНТАЦІЇ ДАНИХ



ПОРІВНЯННЯ ПРОПУСКНОЇ ЗДАТНОСТІ ФРАГМЕНТАЦІЇ ДАНИХ



ПОРІВНЯННЯ ПРОПУСКНОЇ ЗДАТНОСТІ ДЕДУБЛІКАЦІЇ ДАНИХ



НАУКОВА НОВИЗНА

Вперше було розроблено модифікований метод контенто-залежної фрагментації даних Twin CDC, який завдяки двонаправленому пошуку меж фрагментів, використанню двох окремих Gear-таблиць та встановленню альтернативних меж фрагментів забезпечує підвищення пропускної здатності фрагментації даних від 2% до 39%, збільшення пропускної здатності дедублікації від 3% до 46 % та зростання коефіцієнта дедублікації даних на 3 % відносно вже відомих методів фрагментації даних.

ПРАКТИЧНА ЗНАЧУЩІСТЬ

За результатами даного дослідження було виявлено, що запропонована модифікація має кращі кількісні характеристики продуктивності та дедублікації, що робить дану модифікацію придатною для використання у виробничих середовищах, зокрема у системах резервного копіювання, медіа-платформах, а також в інфраструктурі постачальників хмарного постійного сховища.

БІЗНЕС-МОДЕЛЬ

<table>
<tr>
<td rowspan="2">Проблема
Низький ступінь дедублікації, великий час фрагментації даних, високе споживання ресурсів під час фрагментації, постійна потреба у розширенні постійного сховища даних.</td>
<td>Рішення
Модифікований метод Twin CDC та його програмна реалізація, що забезпечує вищий коефіцієнт дедублікації, а також вищу пропускну здатність фрагментації та дедублікації даних.</td>
<td rowspan="2">Унікальна ціннісна пропозиція
Готове рішення контенто-залежної фрагментації даних, що легко інтегрувати у вже існуючу інфраструктуру.</td>
<td>Прихована перевага
Вбудоване середовище тестування та апробації дозволить виконувати тонке налаштування фрагментації під конкретний випадок використання.</td>
<td rowspan="2">Споживачі
Провайдери хмарних сховищ та сервісів резервного копіювання, великі підприємства та корпорації з власними дата-центрами, розробники систем зберігання даних та резервного копіювання, DevOps-інженери та SRE-фахівці, бізнес-керівники та фінансові менеджери замовників</td>
</tr>
<tr>
<td>Ключові метрики
Кількість проданих ліцензій, кількість поновлених ліцензій, відгуки та пропозиції користувачів.</td>
<td>Канали
Через відділ продажів, маркетингові кампанії (реклама), виступи та демо на конференціях з розробки ПЗ.</td>
</tr>
<tr>
<td colspan="3">Структура витрат
Утримання штату (заробітна плата, бонуси), податкове навантаження, подальші розробка та дослідження, маркетингові кампанії та промоутинг, хмарна інфраструктура.</td>
<td colspan="2">Потоки доходу
Продаж ліцензій різних типів та їх поновлення, підтримка споживачів, технічний супровід та аудит.</td>
</tr>
</table>

АПРОБАЦІЯ

Основні положення і результати роботи були представлені у роботі «Модифікований метод та програмне забезпечення для контенто-залежної фрагментації даних» та обговорювалися на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг 2025», що проходила з 19 по 21 листопада 2025 року у м. Київ.

ПОДАЛЬШІ НАПРЯМКИ РОБОТИ

- Паралелізація двонаправленого пошуку
- Оптимізація IO операцій
- Реалізація статично-компільованою мовою програмування

ВИСНОВКИ

1. Проаналізовано відомі методи контенто-залежної фрагментації даних.
2. Запропоновано модифікований метод фрагментації, ключовими особливостями якого є двонаправлений пошук меж, використання двох Gear-таблиць та встановлення альтернативних меж фрагментів.
3. Розроблено програмний комплекс для емпіричного порівняння кількісних метрик запропонованого методу з вже відомими.
4. Запропонований метод має вищий коефіцієнт дедублікації з перевагою до 3%, вищу пропускну здатність фрагментації до 39% та вищу пропускну здатність дедублікації до 46%.
5. Наведено потенційні напрямки роботи для подальшого підвищення продуктивності фрагментації та дедублікації.
6. Сформовано бізнес-модель стартап-проєкту.

Дякую за увагу!

Питання?