УДК 004.9

Б. П. Русин[1], Л. В. Погрелюк[1], В. А. Висоцька[2], М. М. Осипов[2],
Я. Ю. Варецький[1], О. В. Капшій[1]

АРХІТЕКТУРА СИСТЕМИ ДЕДУБЛІКАЦІЇ ТА РОЗПОДІЛУ ДАНИХ У ХМАРНИХ СХОВИЩАХ ПІД ЧАС РЕЗЕРВНОГО КОПІЮВАННЯ

[1]Фізико-математичний інститут імені Г.В. Карпенка НАН України, Львів
[2]НУ «Львівська політехніка», кафедра «Інформаційні системи та мережі», Львів

Анотація. Розроблена та детально описана концептуальна модель системи. Розроблена інтелектуальна система дедуплікації та розподілу даних у хмарному сховищі, описано опис програмного забезпечення, розглядаються етапи роботи користувача. Було проведено тестування роботи спроектованої системи. Описано кілька контрольних зразків, проаналізовано результати. Метою системи є дедублікації та розподілу даних у хмарних сховищах таким чином, щоб в кінцевому результаті резервного копіювання даних усунути повторюючі частини даних використовуючи потужності розподіленого обчислення та хмарних сховищ. Підібравши правильний підхід до розподілення завдань і даних під час дедублікації, можливо використати весь потенціал хмарних розподілених систем для збільшення швидкості резервного копіювання та його пропускної здатності. Проаналізовано (наведено недоліки та переваги використання різних підходів) та обрані ефективні методи вирішення задач: гібридну дедублікацію на рівні блоків, розбиття потоку даних на основі цифрового відбитку Рабіна, розподіл даних на основі хеш значень блоків дедублікації та використання розподіленого індексу. Дедублікація на рівні блоків передбачає два типи розбиття потоків даних на блоки, це розбиття з фіксованою довжиною та змінною на основі алгоритму. Розбиття з фіксованої довжиною досить тривіальне і швидке відносно складності алгоритму, проте недоліком є зміщення даних на початку потоку, оскільки блоки які будуть слідувати після змін будуть вважатися як нові. Проте у випадку з розбиттям блоків зі змінною довжиною, точку власне розбиття визначає алгоритм. Даний алгоритм повинен працювати з безкінечними потоками даних використовуючи кільцеву хеш функцію. Алгоритм поглинає кожен вхідний байт даних з потоку, і як тільки значення кільцевої хеш функції відповідає заданому раніше шаблону, це і слугує точкою розбиття потоку на блоки. Таким чином, при змінені або зміщені даних на пару байтів, новим буде вважатись лише той блок даних який охоплює дані. Проте для того щоб відслідковувати зміни і правильно виставляти точки розбиття необхідно перевіряти вхідні дані на певний заданий цифровий шаблон – хеш значення. Поширеною практикою є обчислення хеш значення кожен раз на отримання вхідного байту в потоці даних. Точкою розбиття стане той момент, коли отримане хеш значення відповідатиме заданому шаблону. Щоб робити такі обчислення ефективно, було придумано алгоритм кільцевого хешу. Один із найпоширеніших алгоритмів кільцевого хешу є цифровий відбиток Рабіна. В ході аналізу засобів вирішення задач було обрано мову програмування Rust для написання клієнтської частини, мову програмування Scala для серверної частини, інструментарій Akka для менеджменту розподілених обчислень та Amazon S3 в якості хмарного сховища.

Ключові слова: дедублікація даних, розподіл даних, хмарне середовище, cloud computing, алгоритм Рабіна, хешування даних, гібрідна дедублікація.

Аннотация. Разработана и подробно описана концептуальная модель системы. Разработана интеллектуальная система дедупликации и распределения данных в облачном хранилище, описано описание программного обеспечения, рассматриваются этапы работы пользователя. Было проведено тестирование работы спроектированной системы. Описаны несколько контрольных образцов, проанализированы результаты. Целью системы является дедубликации и распределения данных в облачных хранилищах таким образом, чтобы в конечном итоге резервного копирования данных усунуть повторяющиеся части данных, используя мощности распределенного вычисления и облачных хранилищ. Подобрав правильный подход к распределению задач и данных при дедубликации, возможно использовать весь потенциал облачных распределенных систем для увеличения скорости резервного копирования и его пропускной способности. Проанализированы (приведены недостатки и преимущества использования различных подходов) и выбранные методы решения задач: гибридную дедубликацию на уровне блоков, раз-биение потока данных на основе цифрового отпечатка Рабина, распределение данных на основе хэш значений блоков дедубликации и использования распределенного индекса. Дедубликация на уровне блоков предусматривает два типа разбиения потоков данных на блоки, это разбиение с фиксированной длиной и переменной на основе алгоритма. Разбивка с фиксированной длиной достаточно тривиальное и быстрое относительно сложности алгоритма, однако недостатком является смещение данных в начале потока, поскольку блоки, которые будут следовать после изменений, будут считаться как новые. Однако в случае с разбивкой блоков с переменной длиной, точку собственно разбиение определяет алгоритм. Данный алгоритм должен работать с бесконечными потоками данных, используя кольцевую хэш-функцию. Алгоритм поглощает каждый входной байт данных из потока, и как только значение кольцевой хэш-функции соответствует заданному ранее шаблону, это и служит точкой разбиения потока на блоки. Таким образом, при изменении или смещены данных на пару байтов, новым будет считаться только тот блок данных, который охватывает данные. Однако для того чтобы отслеживать изменения и правильно выставлять точки разбиения необходимо проверять входные данные на определенный заданный цифровой шаблон - хэш значения. Распространенной практикой является вычисление хэш значение каждый раз на полу-ния входного байта в потоке данных. Точкой разбиения станет тот момент, когда полученное хеш значение будет соответствовать заданному шаблону. Для таких вычислений эффективно использовать алгоритм кольцевого хеша. Один из самых распространенных алгоритмов кольцевого хеша является цифровой отпечаток Рабина. В ходе анализа средств решения задач был выбран язык программирования Rust для написания клиентской части, язык программирования Scala для серверной части, инструментарий Akka для менеджмента распределенных вычислений и Amazon S3 в качестве облачного хранилища.

Ключевые слова: дедубликация данных, распределение данных, облачную среду, cloud computing, алгоритм Рабина, хеширования данных, гибридная дедубликация.

Abstract. The conceptual model of the system is developed and described in detail. An intelligent system of deduplication and distribution of data in the cloud storage is developed, the description of the software is described, the stages of the user's work are considered. Testing of the projected system was carried out. Several control samples are described and results are analyzed. The purpose of the system is to deduplicate and distribute data in cloud repositories in such a way that the end result of the backup is to eliminate duplicate pieces of data using

distributed computing and cloud repositories. By picking the right approach to distribute tasks and data during deduplication, you can harness the full potential of cloud-based distributed systems to increase backup speed and bandwidth. Analyzes (disadvantages and advantages of using different approaches) are analyzed and effective methods of solution are selected: hybrid block-level deduplication, splitting of data flow on the basis of Rabin's digital imprint, distribution of data based on hash values of blocks of deduplication and use of distributed index. Block-level deduplication involves two types of data flow splitting into blocks, a fixed-length, algorithm-based split. Fixed-length partitioning is rather trivial and fast with respect to the complexity of the algorithm, but the downside is that data is displaced at the beginning of the stream, since the blocks that will follow after the changes will be considered new. However, in the case of partitioning of blocks of variable length, the point of proper partitioning is determined by the algorithm. This algorithm should work with infinite data flows using the ring hash function. The algorithm absorbs each input byte of data from the stream, and as soon as the value of the annular hash function corresponds to the previously specified template, it also serves as a point of splitting the stream into blocks. Thus, if the data is changed or displaced by a couple of bytes, only the data block that covers the data will be considered new. However, in order to track changes and correctly set breakpoints, it is necessary to check the input data for a specific preset digital pattern - a hash value. It is a common practice to calculate a hash value every time an input byte is received in a data stream. The point of partition will be the moment when the resulting hash value matches the specified pattern. To do these calculations effectively, an algorithm has been devised for the ring hash. One of the most common ring hash algorithms is a digital Rabin imprint. During the analysis of the solutions, the Rust programming language for client-side writing, the Scala programming language for the server-side, the Akka distributed computing management tool, and Amazon S3 as the cloud repository were selected.

Key words: data deduplication, data sharing, cloud environment, cloud computing, Rabbin algorithm, data hashing, hybrid deduplication.

DOI: https://doi.org/10.31649/1999-9941-2019-45-2-40-63.

Вступ

Для стабільної роботи компаній які працюють з інформацією є процес резервного копіювання даних. Не залежно від того, чи це банк, чи сайт по продажам господарських товарів, у разі несистемного збою та пошкодження основних серверів або баз даних – в найкоротший термін, система помилка повинна буде виправлена а резервні дані відновлені. Проте періодичне резервне копіювання викликає проблему з повторюваними даними, тим самим створюючи проблему для відділів фінансів в зв’язку з великими витратами на послуги постачальників хмарних сховищ або власних дата-центрів. Для економії дискового простору або коштів на послуги хмарних сховищ, багато систем резервного копіювання використовують дедублікацію даних [1-2]. Даний підхід дозволяє зменшити кінцевий розмір резервних копій, шляхом відсічення дублюючих даних або тих що уже відомі системі (були раніше збереженні). Проте дані системи використовують хмарні технології лише в якості кінцевих точок сховищ. Підібравши правильний підхід до розподілення завдань і даних під час дедублікації, можливо використати весь потенціал хмарних розподілених систем для збільшення швидкості резервного копіювання та його пропускної здатності. З огляду на те, актуальною задачею є дослідження та подальша розробка інтелектуальної системи дедублікації та розподілу даних у хмарних сховищах.

Актуальність

На основі того де відбувається розбиття потоку даних на блоки, обчислення їх хешів, побудова індексу та зберігання унікальних блоків, можна виділити такі типи дедублікації: вихідна, цільова, потокова та гібридна.

Вихідна дедублікація (англ. *source deduplication*) – це дедублікація яка виконується на пристрої, де знаходяться вихідні дані [3]. Будь-які дані, які помічені для резервного копіювання, будуть поділені на блоки, для яких пораховані хеші. На основі отриманих хешів, буде побудований індекс для подальшої дедублікації даних. Проте у даного підходу є два потенційні недоліки.

Перший потенційний недолік полягає в тому, що для процесу дедублікації використовуються ресурси девайсу, на якому знаходяться дані. Таким чином, користувачі повинні переконатися, що у них є достатньо вільної оперативної пам'яті та обчислювальної потужності процесора. А отже, резервне копіювання, наприклад поштового сервера або інтернет веб-сайту може призвести, щонайменше, до уповільнення їх роботи.

Другий недолік стосується розташування хеш-таблиці, тобто індексу дедублікації. У деяких випадках хеш-таблиця обмежується кожним окремим комп'ютером, який виконує дедублікацію. В такому випадку, індекс буде обмеженим лише локальними даними. Тобто переваги дедублікації будуть обмежені лише в межах одного комп'ютер. Якщо на інших серверах існують ідентичні фрагменти даних, поточний сервер не знатиме цього.

Цільова дедублікація (англ. *target deduplication*) – це дедублікація яка виконується на цільовому центральному сервері, де на кінцевому етапі будуть зберігатись дані [3]. Як тільки дані прибудуть до репозиторія, сервер отримає можливість побудувати індекс на їх основі та зробити перетин з існуючою, центральною хеш-таблицею, щоб зберегти лише унікальні дані. Перевагою даного підходу є великий об'єм даних. А оскільки чим більше даних, тим більший індекс, що в результаті дає більше шансів знайти ідентичні блоки, збільшуючи коефіцієнт дедублікації. Також вагомою перевагою є перенесення відповідальності за обчислювальні потужності від клієнта (пристрою з вихідними даними) на цільовий сервер, який розрахований лише на дедублікацію, унеможливлюючи потенційну боротьбу за обчислювальні ресурси. Проте даний варіант не вирішує всі проблеми, є певні моменти які необхідно розглянути

і взяти до уваги. Цільова дедублікація вимагає щоб дані були збережені на диск перед самим процесом розбивки і хешування, щоб не перевантажувати клієнтів. А отже слабким місцем даного підходу є швидкість запису/читання з диску.

Другим потенційним недоліком є можливість колізій хеш-функцій, тобто можливість того, що два блоки даних які не є ідентичні, отримають однакове хеш значення. Хеш колізії можуть призвести до втрати консистентності даних, що у кінцевому випадку пошкодить оригінальні дані без можливості їх відновлення. Щоб уникнути колізії необхідно використовувати алгоритми, в яких шанс їх утримання менший, проте це призводить до більшого навантаження на обчислювальні ресурси. Проте, використання більш сильних та складних алгоритмів не є проблемою, оскільки сервери дедублікації використовують спеціалізоване апаратне обладнання під необхідні задачі.

Третім недоліком є те що повний об'єм даних для резервного копіювання повинен бути переданий через мережу від клієнтів до центрального сховища. Дана проблема є особливо актуальна якщо у клієнта наявне слабке або високо тарифіковане мережеве покриття. При незначних змінах після останнього резервного копіювання, надсилання абсолютно всіх даних є надлишковим. Також, проблемою може виникнути завантаженість мережі цільового сервера, що негативно вплине на кінцеву пропускну здатність.

Потокова дедублікація (англ. *inline deduplication*) використовує оперативну пам'ять в якості буферної зони для процесу розбивки даних і обрахування їх хешів [4]. А отже дані дедублюються перед записом на диск, це виключає витрати на швидкість запису/читання з диску. Потокову дедублікацію варто розглядати, як кращу версію цільової дедублікації. Перша, має всі переваги останньої, проте жодного недоліку пов'язаного з повільною роботою диску.

Гібридна дедублікація – це підхід при поєднанні вихідної та цільової (потокової) дедублікації[4]. Розбиття даних на блоки і обрахунок їх хешів відбувається на клієнті (пристроїв з вихідними даними). Отримані хеші надсилаються на сервер для їх ідентифікації, після чого, клієнт надсилає серверу лише ті блоки даних, які раніше йому не відомі.

Даний підхід нівелює недолік цільової та потокової дедублікації щодо використання інтернет мережі. Оскільки з клієнта буде надіслано лише ті блоки даних які раніше невідомі, а у випадку уже відомих, надсилаються лише їх хеші, які 2-3 тис. разів менші за оригінальні дані.

Хоча клієнт, у випадку з гібридною дедублікацією, все ще бере участь у розбивці даних на блоки та обчислення їх хешів, як і у випадку з вихідною дедублікацією. Проте робота з індексом лежить повністю на цільовому репозиторії, що значно розвантажує обчислювальні потужності клієнта. Якщо, даний момент все ще є недоліком для того чи іншого випадку, то розбивку даних на блоки та обчислення їх хешів можливо перенести на інший, транзитний пристрій який знаходиться всередині мережі клієнта або й повністю делегувати цю роботу до сервера. Дедублікація на рівні блоків передбачає два типи розбиття потоків даних на блоки, це розбиття з фіксованою довжиною та змінною на основі алгоритму [4]. Розбиття з фіксованої довжиною досить тривіальне і швидке відносно складності алгоритму, проте недоліком є зміщення даних на початку потоку, оскільки блоки які будуть слідувати після змін будуть вважатися як нові. Проте у випадку з розбиттям блоків зі змінною довжиною, точку власне розбиття визначає алгоритм. Даний алгоритм повинен працювати з безкінечними потоками даних використовуючи кільцеву хеш функцію. Алгоритм поглинає кожен вхідний байт даних з потоку, і як тільки значення кільцевої хеш функції відповідає заданому раніше шаблону, це і слугує точкою розбиття потоку на блоки. Таким чином, при змінені або зміщені даних на пару байтів, новим буде вважатись лише той блок даних який охоплює дані. Проте для того щоб відслідковувати зміни і правильно виставляти точки розбиття необхідно перевіряти вхідні дані на певний заданий цифровий шаблон – хеш значення. Поширеною практикою є обчислення хеш значення кожен раз на отримання вхідного байту в потоці даних. Точкою розбиття стане той момент, коли отримане хеш значення відповідатиме заданому шаблону. Щоб робити такі обчислення ефективно, було придумано алгоритм кільцевого хешу [4-5]. Один із найпоширеніших алгоритмів кільцевого хешу є цифровий відбиток Рабіна [4].

Мета

Метою роботи є розробити загальну архітектуру системи дедублікації та розподілу даних у хмарних сховищах під час резервного копіювання.

Задачі

1. Здійснити аналіз сучасних технологій проектування та створення систем у сфері резервного копіювання та відновлення даних, здійснити порівняльний аналіз технічних реалізацій та алгоритмів, застосованих в реалізаціях.
2. Побудувати концептуальну модель інтелектуальної інформаційної системи дедублікації та розподілу даних у хмарних сховищах, задля автоматизації та ефективної роботи ‘програмного забезпечення у сфері резервного копіювання та відновлення даних.

3. Провести аналіз методів та засобів реалізації інтелектуальної системи та зробити вибір для реалізації власної.
4. Спроектувати інтелектуальну систему дедублікації та розподілу даних у хмарних сховищах.

Розв'язання задач

Проектування інтелектуальної системи дедублікації та розподілу даних у хмарних сховищах це складний процес, що потребує детального опису та ставить перед дослідниками множину питань. Оскільки об'єктом дослідження є процес дедублікації та розподілу даних у хмарних сховищах, важливим є на даному етапі провести системний аналіз обраної предметної області, для того, щоб розуміти повну картину роботи системи, процеси перебігу даних в цій системі, визначити головні можливості системи та основні бізнес процеси для того, щоб на етапі реалізації уникнути багатьох помилок в архітектурному плані та виборі моделей вирішення задачі дедублікації та розподілу даних у хмарних сховищах. Важливим є аналіз мети системи і розуміння шляхів її досягнення, застосувавши наявні ресурси - технічні засоби, певні методи та підходи, архітектури для побудови інформаційної системи. Системний аналіз дозволить розділити поставлену перед нами задачу на сукупність простих задач, враховуючи взаємозв'язки простіших задач між собою [6-10]. Таким чином складна система розбивається на складові та з чітко формуються бізнес процеси між частинами продукту. Конкретизація принципів системного підходу до аналізу інтелектуальної системи дедублікації та розподілу даних у хмарних сховищах дає краще уявлення та бачення системи загалом, розуміння її функціонування, необхідних кроків для вдосконалення та певних особливостей системи. Метою даного дослідження є розробка програмного забезпечення, що матиме змогу проводити дедублікацію та розподілу даних у хмарних сховищах. Перш за все, проектовану систему необхідно розглядати як множину більш простих елементів - її складових, які є окремими модулями між якими відбувається постійна комунікація, тобто елементи є взаємопов'язані. Потрібно також розуміти і те, що як і всередині системи відбуваються певні процеси, так і зовні на систему впливають фактори, які також вносять свою частину у функціонування системи - за допомогою цих факторів також відбувається перетворення потоків даних. Тож, можемо визначити основні принципи системного підходу при побудові інтелектуальної системи дедублікації та розподілу даних у хмарних сховищах [11]:

- принципом кінцевої мети є дедублікації та розподілу даних у хмарних сховищах;
- принцип єдності полягає у тому, що система розглядається як сукупність певних складових, але з іншого боку - проектоване програмне забезпечення є цілісною системою, оскільки функціональні процеси, якими вона володіє, логічно випливають з елементів, що формують систему, проте жоден елемент окремо не володіє цими функціями;
- принципом зв'язності підтверджується функціонуванням системи в певному середовищі - очевидно, що інтелектуальна система дедублікації та розподілу даних у хмарних сховищах буде корисною у використанні багатьма компаніями в задачі резервного копіювання даних, відповідно, необхідно передбачити механізм гнучкого налаштування для різних підходів;
- принцип модульної побудови визначається модульною структурою проекту, тобто повноцінна робоча система дедублікації та розподілу даних у хмарних сховищах поділяється на певну кількість модулів, між якими відбувається комунікація (модулі можуть бути підключеними або відключеними в залежності від потреб клієнтів);
- принцип функціональності визначає, що проектування структури системи можливо лише після визначення та повного розуміння роботи функцій системи, відповідно для початку маємо точно сформулювати та детально оглянути можливості системи дедублікації та розподілу даних у хмарних сховищах;
- принцип розвитку означає, що система має бути спроектована таким чином, щоб пізніше мати можливість розвиватись за рахунок розширення, підключення додаткових модулів, заміною імплементації інтерфейсів, накопичення даних в базах та можливістю масштабування;
- принцип децентралізації - це правильне співвідношення між централізацією та децентралізацією системи, що визначається метою та призначенням системи;
- невизначеності - ймовірність певних непередбачуваних факторів, що можуть вплинути на систему, щось, що може відбутись, а може і не відбутись.

Головним завданням системного аналізу є побудова моделі (системи), що дозволить розв'язати поставлену задачу. Кінцева система повинна складатись з взаємопов'язаними між собою елементами, що в сукупності приводять до вирішення конкретної проблеми. Базу системного аналізу складають система, мета, елемент, функція, оточення та інші. Під поняттям система розуміється сукупність окремих елементів між якими відбувається певна взаємодія, що веде до досягнення певної спільної мети, а також ця множина елементів утворюють цілісну систему, що взаємодіє з навколишнім середовищем як одне ціле [12-17].

Зовнішньою сутністю інтелектуальної системи дедублікації та розподілу даних у хмарних сховищах, що функціонує у вигляді програми резервного копіювання даних є адміністратор, що проводить резервне копіювання даних підприємства, або користувач, що проводить резервне копіювання персональних даних. Елементами системи, що аналізується є: користувацький інтерфейс, вхідна точка для користувачів системи, надає змогу вибирати файли для резервного копіювання, переглядати існуючі копії, управляти їхнім життєвим циклом, розподілом та планувати завдання для періодичного резервного копіювання даних, а також відновлювати дані з резервних копій; програма-клієнт, яка отримує дані від користувацького інтерфейсу, проводить розподіл потоку даних на блоки та обчислення їх значень хеш-функції, тестування хеш значень блоків з індексом системи та надсилання нових блоків даних до хмарних сховищ; індекс - хеш таблиця яка містить інформацію про всі відомі унікальні блоки даних системи, відповідає за усунення дублікатів на момент збереження їх до хмарних сховищ; програма-сервер - виконує алгоритм дедублікації оперуючи блоками даних програми-клієнта та індексом, проводить розподіл завдань дедублікації між робітниками (окремими вузлами в кластерному середовищі) та комунікацію з хмарними середовищами для зберігання даних.

Метою системи є дедублікації та розподілу даних у хмарних сховищах таким чином, щоб в кінцевому результаті резервного копіювання даних усунути повторюючі частини даних використовуючи потужності розподіленого обчислення та хмарних сховищ. Важливим буде зрозуміти та визначити середовище функціонування системи, що означатиме вибір такої множини об'єктів, на яку мала би вплив розроблювана система і яка б змінювалась під дією цього впливу, а також такі, чия зміна впливала б на проектовану систему [18-21]. Взаємодія інформаційної системи з зовнішнім середовищем відбувається завдяки наявним входам та виходам системи. Вплив зовнішнього середовища на проектовану систему є, власне, входом, а результат цього впливу є виходом. Зазвичай прийнято розглядати лише найважливіші зв'язки системи з зовнішнім середовищем, в даному випадку це буде кінцевий користувач, що приймає участь в процесі резервного копіювання даних та певне підприємство, що використовуватиме розроблюваний програмний продукт та забезпечуватиме вхідні дані для проектованої системи.

Відомо, що в методології системного аналізу системи діляться на два типи: складні та прості. Системи, що складаються з невеликої кількості елементів та, відповідно, зав'язків між ними, не мають розгалуженої структури називаються простими системами. Складними ж є ті, що побудовані з значно більшої кількості елементів, відповідно мають більшу кількість внутрішніх зав'язків, виконують складні функції та є неоднорідними [22-27]. Очевидно, що програмний продукт, що проектується, є представником складних систем. Проектована інтелектуальна система дедублікації та розподілу даних у хмарних сховищах складається з великої кількістю елементів, матиме потужні канали комунікації між ними, ефективність роботи кожного з елементів окремо буде значно нижчою ніж ефективність використання їх сукупності у вигляді цілої системи.

Діаграма варіантів використання (англ. *use case diagram*) або діаграма прецедентів - це діаграма UML, на якій зображено відношення між акторами (зовнішні сутності) та прецедентами (варіанти використання) в системі [28-32]. Діаграма є графом, що складається з множини акторів, прецедентів обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. На діаграмі варіантів використання (рис. 1) також зображено наступні прецеденти (варіанти використання):

- *Надіслати дані* – користувач формує список файлів і надсилає їх до системи для резервного копіювання.
- *Розбити дані на блок* – система формує потік з вхідних даних і розбиває їх на маленькі блоки даних для комфортної дедублікації.
- *Обчислити хеші блоків* – система для кожного вхідного блоку обчислює хеш значення використовуючи задані хеш-функції.
- *Усунути дублікати блоків* – використовуючи *Індекс,* система перевіряє існуючість блоків за допомогою отриманих хеш значень.
- *Розподілити блоки між робітниками* – для збільшення пропускної здатності та швидкодії, система розподіляє роботу на блоками між робітниками в системі (наприклад вузли в кластері).
- *Зберегти унікальні блоки* – після усунення дублюючих блоків, система, використовуючи *Хмарні сховища*, зберігає унікальні блоки в сховищах.
- *Стиснути блоки* – для економії дискового простору, вхідні блоки можуть бути стиснуті один із алгоритмів компресації.
- *Зашифрувати блоки* – для додаткової захищеності даних, вхідні блоки можуть бути також зашифровані.
- *Сформувати резервну копію* – після закінчення обробки вхідних блоків, вихідний потік даних зберігається як послідовність посилань на хеш значення.

- *Отримати резервну копію* – після закінчення процесу дедублікації, користувач отримує посилання на резервну копію для майбутнього відновлення даних
- *Відновити дані* – використовуючи посилання на резервні копії, користувач може відновити оригінальні дані.

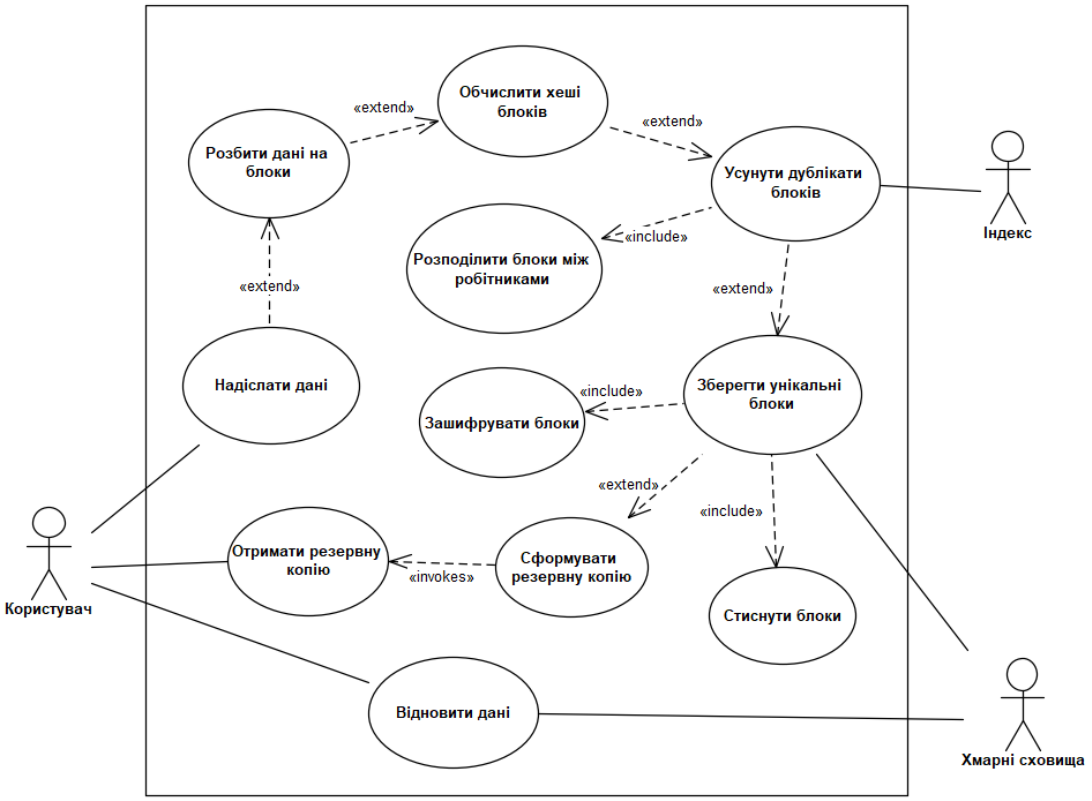

Рисунок 1 – Діаграма варіантів використання системи

Діаграма станів – це UML діаграма, що визначає зміну станів об'єкта у часі [15]. Діаграма діяльності описує як процеси виконання певних робіт, виконувані або системою або зовнішніми сутностями координують між собою. Виконання певних робіт може бути залежним від деяких передумов, що мають бути виконані. Перехід до іншої дії відбувається після завершення виконання попередньої. Зазвичай дія на вході отримує певну інформацію - множину сигналів, що має бути опрацьована і перетворена на множину вихідних сигналів. На рис. 2 подано діаграму станів резервного копіювання даних користувачем інтелектуальної системи дедублікації та розподілу даних у хмарних сховищах. Об'єктом діаграми станів є процес резервного копіювання, і дальше в цьому підпункті ми будемо розуміти цей процес під назвою об'єкт. Стан *автентифікація користувача* є початковим для процесу резервного копіювання. Якщо дані автентифікації введено неправильно то об'єкт залишається в тому самому стані і очікує на нові команди. Якщо дані автентифікації, а саме логін та пароль, введено правильно то об'єкт переходить у стан *вибору файлів для резервного копіювання.* Наступний стан – *створення потоку даних.* З вибраних файлів для дедублікації система створює єдиний потік даних. Після успішного створення потоку даних, об'єкт переходить у стан *розбиття потоку даних на блоки*. Блок даних – це одиниця процесу дедублікації, саме дублікати блоків система старається уникати. Якщо в потоці даних є дані для розбиття на блоки, то об'єкт переходить у стан *обчислення хеш значення блоку*. Кожен блок даних повинен мати свій цифровий підпис, який набагато менший по обсягу, чим і являється хеш значення. Після того як отримано хеш значення, об'єкт переходить у стан *перевірки хеш значення на існуючість.* Для того щоб відкинути дублікати, треба перевірити чи даний блоку існує в системі, проте перевіряти весь блок цілком витратно, тому перевіряють на існуючість їх хеш значення. Якщо хеш значення уже відоме системі, то об'єкт переходить назад у стан *розбиття потоку даних* на блоки. А якщо хеш значення невідоме, то об'єкт переходить у стан *збереження блоку даних.* Новий блоку даних, раніше не відомий системі, зберігається у спеціально відведеному місці в хмарному сховищі, базуючись на його хеші (для спрощення його подальшого пошуку). Після збереження, об'єкт переходить у назад у стан *розбиття потоку даних на блоки.* Коли об'єкт знаходиться у цьому стані і більше не може отримати нових блоків з потоку даних, об'єкт переходить у

стан *побудови резервної копії.* Після процесу дедублікації, список з блоків даних представлений у вигляді списку з його хеш значень, що і являється резервною копією. Після її отримання, об’єкт переходить у стан *збереження резервної копії.* Даний стан є останнім перед входом об’єкт до кінцевого стану.

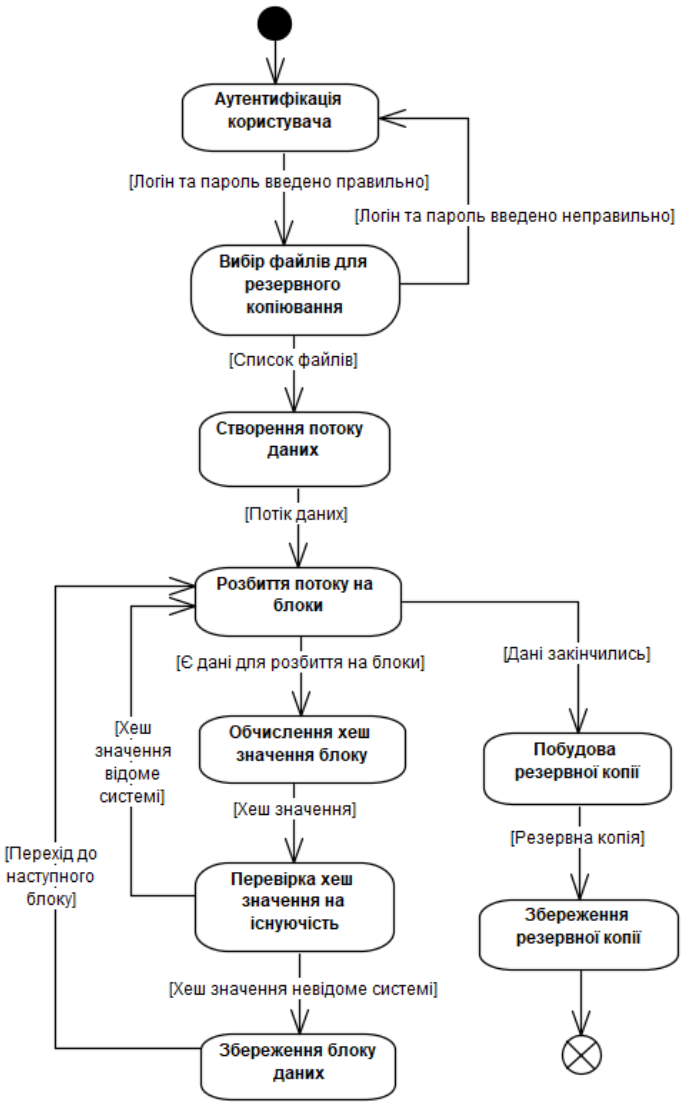

Рисунок 2 – Діаграма станів резервного копіювання

Діаграми послідовностей – це UML діаграма, яка відображає взаємодії об'єктів впорядкованих за часом [15]. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність відправлених повідомлень. На діаграмі послідовностей показано у вигляді вертикальних ліній різні процеси або об'єкти, що існують водночас. Надіслані повідомлення зображуються у вигляді горизонтальних ліній, в порядку відправлення. На рис. 3 подано діаграму послідовності резервного копіювання даних, використовуючи інтелектуальну систему дедублікація та розподілу даних у хмарних сховищах. На діаграмі зображено лінію життя актора *Користувач* та чотирьох об’єктів: *Програма-клієнт, Програма-сервер, Індекс* та *Хмарні сховища.*

- *Користувач* – єдиний актор в системі, обирає дані для резервного копіювання і отримує результат дедублікації.
- *Програма-клієнт* – програмне забезпечення на стороні користувача, яке відповідальне за групування користувацьких даних і формування потоку байтів на їх основі. Керує процесом дедублікації зі сторони користувача.
- *Програма-сервер* – програмне забезпечення на стороні сервера. Центральний координатор процесу дедублікації, виконує спілкування з програмою-клієнтом для отримання списку хешів та блоків; формує список необхідних, нових блоків на основі даних з індексу; вико-

нує запис нових блоків у хмарних сховищах; формує дерево посилання для відновлення вихідних даних.

- *Індекс* – хеш таблиця, містить усі відомі системі хеші та посилання на відповідний блок даних у хмарі чи іншому сховищі.
- *Хмарні сховища* – місце збереження блоків даних та резервних копій (дерева посилань).

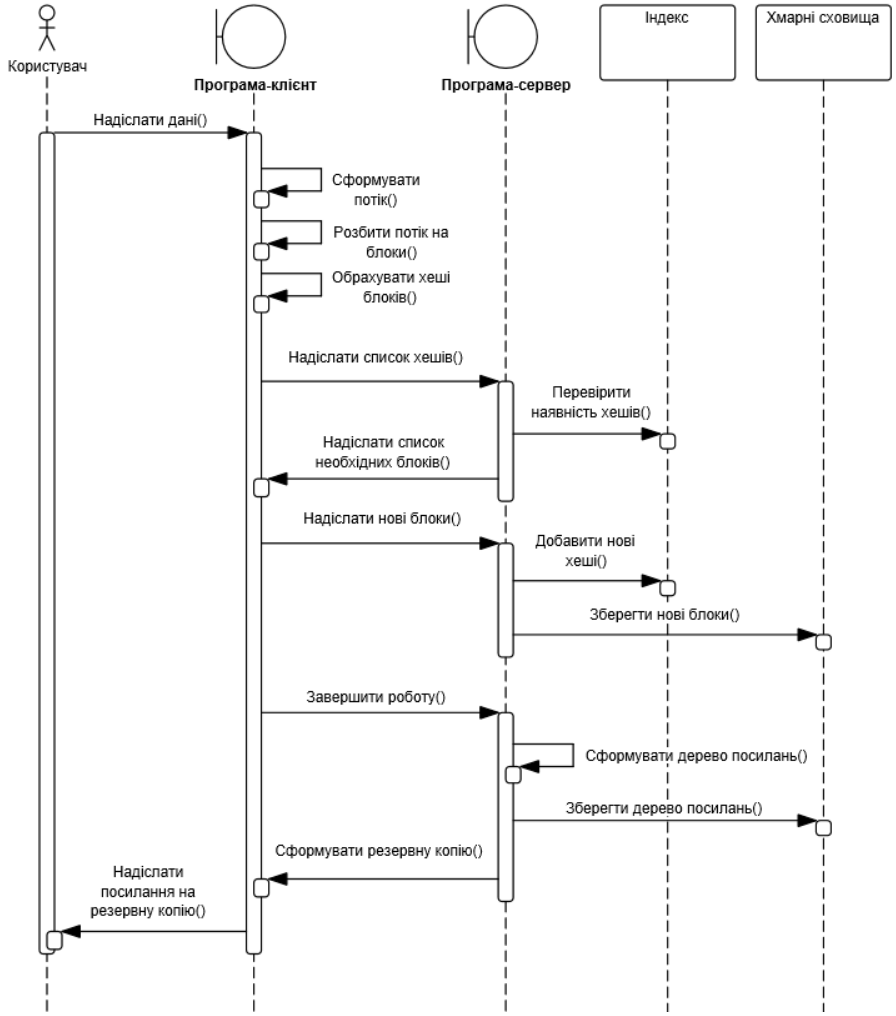

Рисунок 3 – Діаграма послідовності

Загалом робочий процес, згідно діаграми послідовності, виглядає наступним чином. *Користувач*, використовуючи користувацький інтерфейс, у вигляді веб-сайту або настільного додатку, вибирає необхідні файли/дані для дедублікації і надсилає їх до *Програми-клієнта.* Який, в свою чергу, розбиває їх на блоки використовуючи заданий алгоритм. Для кожного блоку *Програма-клієнт* обчислює хеш значення та надсилає їх у вигляді списку до *Програми-сервера.* Останній перевіряє хеші з отриманого списку з даними з *Індексу* та відтинає ті хеші які вже відомі. В результаті назад до *Програми-клієнта* відсилаються лише ті хеші які раніше не відомі системі, а отже яких необхідно добавити в систему. На основі цього списку, *Програма-клієнт* вибірково надсилає блоки до *Програми-сервера*. Кожний отримай блок зберігається в хмарних сховищах з унікальним ідентифікатором (посиланням на блок). Також *Програма-сервер* добавляє в хеш-таблицю *Індексу* нову пару ключ-значення: хеш блоку та посилання на нього. Даний цикл, перевірка на існуючість хеш значень та надсилання нових блоків виконується до тих пір, поки є дані в потоці. Коли потік завершиться, *Програма-клієнт* сигналізує про завершення процесу дедублікації. *Програма-сервер* на основі обробки блоків даних та їх хешів будує резервну копію у вигляді дерева посилань на хеші. Сформована копія повертається до *Користувача*.

Окрім поведінкових діаграм UML, необхідно зобразити систему використовуючи структурні діаграми [15]. Першою структурною діаграмою UML – це діаграма компонентів. На ній зображують компоненти та зв'язки між ними. Компоненти з'єднуються між собою за допомогою структурних

зв’язків (англ. *assembly connector*). Дані зв’язки встановлюються між інтерфейсами двох компонентів, при чому компоненти зазвичай мають як мінімум один інтерфейс. В такому зв’язку двох компонентів, один із них надає послуги необхідні іншому, за шаблоном клієнт-сервер. Також, компоненти можуть мати ієрархію, тобто один компонент може містити в собі інший.

На рис. 4 подано діаграму компонентів інтелектуальної системи дедублікації та розподілу даних у хмарних сховищах, яка містить дві основні компоненти – *Client* та *Server*.

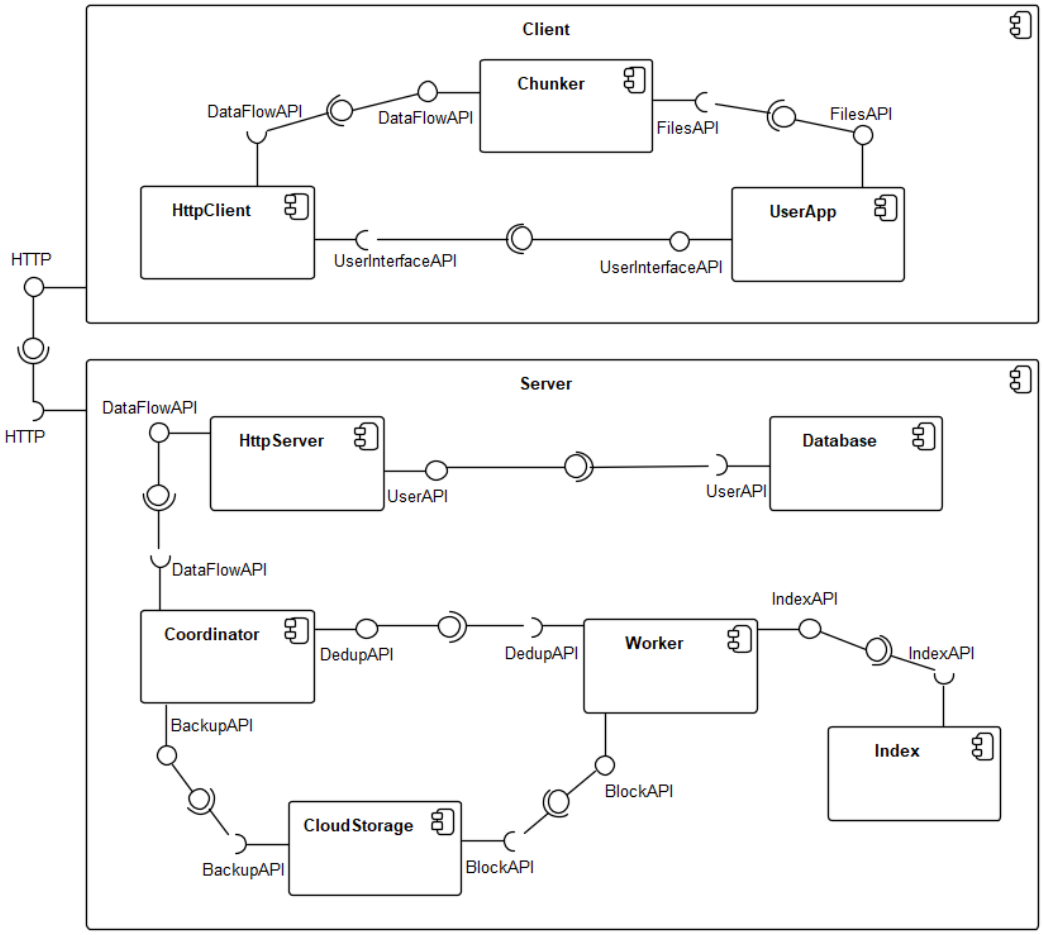

Рисунок 4 – Діаграма компонентів

Компонента *Client* містить програмні компоненти які знаходяться на стороні користувача:

- *UserApp* – компонент користувацької програми, яка надає змогу користувачу автентифікуватись в системі, вибрати файли для резервної копії, запустити сам процес резервного копіювання та відновити дані з резервної копії. Згруповані дані для резервної копії передає до компоненти *Chunker* через інтерфейс *FilesApi,* решта операцій зв’язаний зі взаємодією із сервером – через інтерфейс *UserInterfaceApi* із компонентною *HttpClient*.
- *Chunker* – компонент формування та розбиття потоку даних, який відповідає за перетворення вхідних користувацький файлів (отриманих від *UserApp* через інтерфейс *FilesApi*) у єдиний потік байтових даних, розбиття його на блоки даних та відправку цих блоків до *HttpClient* через інтерфейс *DataFlowApi*.
- *HttpClient* – компонент HTTP-клієнта, приймає дані та команди від користувацького інтерфейсу компоненти *UserApp* (через *UserInterfaceApi*), а також отримання блоків дедублікації від компоненти *Chunker* (через *DataFlowApi*). Усі отримані запити передаються на сервер через HTTP.

Компонента *Server* містить програмні компоненти які знаходяться на стороні сервера:

- *HttpServer* – компонент HTTP-сервера, який обробляє HTTP запити від компоненти *Client*. Вхідні запити пов’язані з процесом резервного копіювання передаються до компоненти *Coordinator* (через *DataFlowApi*), а ті що з процесами користувача (автентифікація користувача) до компоненти *Database* (через UserApi).

- *Database* – компонент бази даних, який відповідає за збереження та читання даних користувача та його резервних копій; а також відповідає за процес автентифікації, перевіряючи отримані логін та пароль зі записами в базі даних.
- *Coordinator* – компонент координатора системи, керує процесом резервного копіювання та дедублікації даних. Розподіляє вхідні блоки дедублікації між робітниками системи (компоненти Worker) та комунікує між ними через інтерфейс *DedupApi*. По завершенню процесу дедублікації, відповідає за формування резервних копій та збереження їх у хмарних сховищах (компонента *CloudStorage)*, використовуючи *BackupApi.*
- *Worker* – компонент робітників системи, які відповідальні за процес дедублікації даних, а саме відсічення дублюючих блоків та збереження нових. Спілкується з компонентом *Index* (через інтерфейс *IndexApi*) для перевірки хеш значень блоків на існуючість у системі, а також збереження хеш значень нових блоків, які власне зберігаються у системі у хмарних сховищах, використовуючи компоненту *CouldStorage* через їх спільний інтерфейс *BlockApi.*
- *CloudStorage* – компонент хмарних сховищ, відповідає за збереження сформованих файлів з блоками даних та деревами посилань резервних копій у хмарних сховищах.
- *Index* – компонент, який відповідає за формування індексу дедублікації та перевірки хеш значень блоків на існуючість.

Діаграми класів – це структурна UML діаграма, яка описує структуру системи демонструючи класи системи, їхні атрибути, методи та зв'язок з іншими класами [15]. Діаграма класів це наріжний камінь моделювання об’єктно-орієнтованих програм. Використовується для загального концептуального моделювання структури програмного забезпечення. Класи в діаграмі класів представляють основні елементи програми, їхній взаємозв’язок в програмі та власне самі класи які повинні бути запрограмовані. У самих діаграмах класи зображені як прямокутники з трьома відсіками:

- ім’я класу, з великою букви, жирним шрифтом по середині відсіку;
- атрибути класу, з малої букви, вирівняні по лівому краю;
- методи класу, з малої букви, вирівняні по лівому краю.

Якщо класи зв’язані між собою то між ними проводить зв'язок асоціації. Якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарною. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів; вони називаються n-арнимі. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами. На рис. 5 подано діаграму класів досліджуваної інтелектуальної системи дедублікації та розподілу даних у хмарних сховищах. Клас *User* представляє користувача системи. Містить інформацію про користувача та відповідає за створення резервних копій та відновлення даних. Атрибути та методи:

- *username*, атрибут типу *String* – унікальний ім’я користувача у інтелектуальній системі, використовується для його ідентифікації.
- *email,* атрибут типу *String* – електронна пошта користувача, використовується під час автентифікації.
- *password,* атрибут типу *String* – пароль користувача, використовується під час автентифікації.
- *auth,* метод з результуючим типом *boolean* – проводить автентифікацію користувача на основі його електронної пошти та паролю.
- *startBackup,* метод з вхідним параметром шляху до файлів для резервного копіювання – розпочинає процес дедублікації. Результатом є об’єкт типу *Stream,* який представляє потік даних для дедублікації.
- *getBackup,* метод з аналогічним вхідним параметром що і у *startBackup* – перевіряє чи дана резервна копія завершилась, і якщо так, то повертає об’єкт типу *Backup.*
- *recover,* метод зі вхідними параметрами об’єкту резервного копіювання типу *Backup* та шляху до відновлених файлів типу *String* – відновлює оригінальні дані з резервної копії у вказаних шлях на диску комп’ютера.

Клас *Backup* представляє резервну копію, яка містить інформацію про її час створення, оригінальний розмір, розмір після дедублікації та власника. Відповідає за відновлення оригінальних даних. Атрибути та методи:

- *id,* атрибут типу *String* – унікальний ідентифікатор резервної копії.
- *timestamp,* атрибут типу *DateTime* – час створення резервної копії.
- *originalSize,* атрибут типу *long* – оригінальний розмір даних у резервній копії.
- *dedupSize,* атрибут типу *long* – розмір даних резервної копії після дедублікації.

- *user*, атрибут типу *User* – власник резервної копії.
- *recover*, метод – відновлює файли з резервної копії. Результатом є список об’єктів типу *FileStream* – клас, що містить метадані про файл та потік байтів самого файлу.

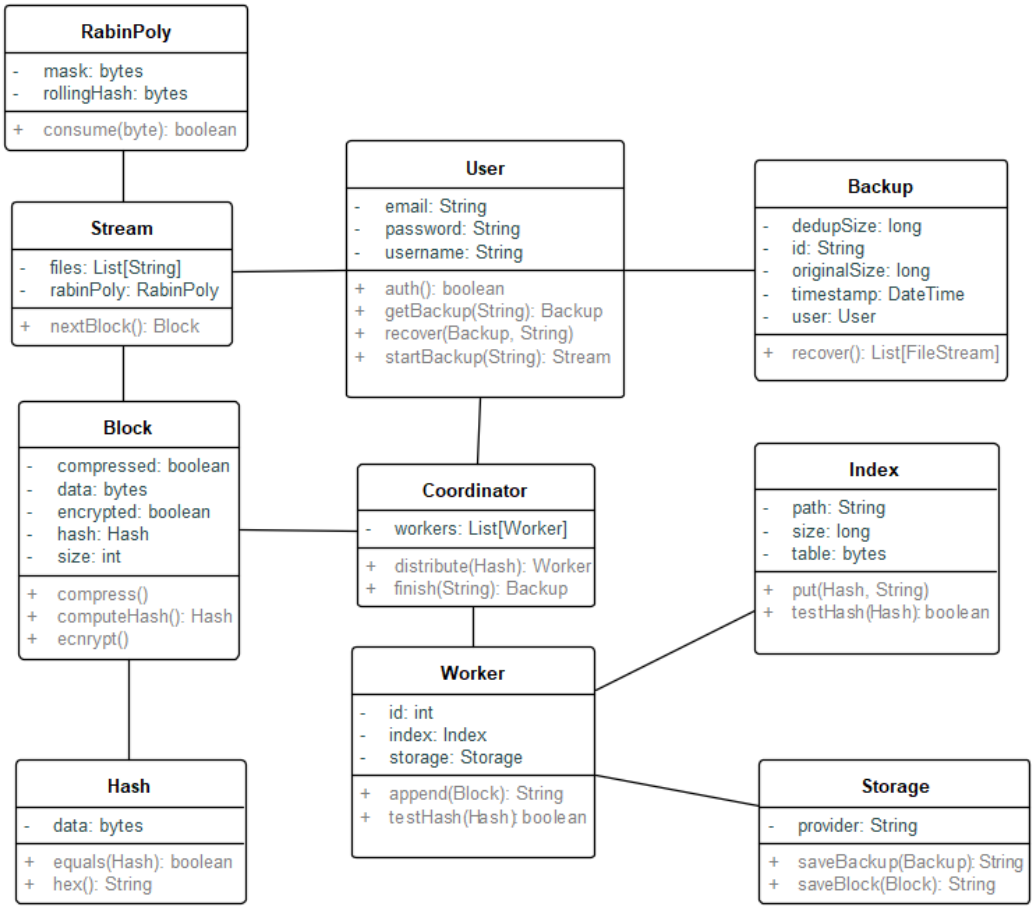

Рисунок 5 – Діаграма класів

Клас *RabinPoly* представляє компонент розбиття потоку даних на блоки використовуючи алгоритм Рабіна. Атрибути та методи:

- *mask*, атрибут типу *bytes* – встановлена маска яка використовується для логічного добутку з кільцевим хешем щоб отримати точку розбиття даних.
- *rollingHash*, атрибут типу *bytes* – кільцевий хеш, змінюється на основі кожного вхідного байту з потоку даних.
- *consume*, метод з вхідним параметром типу *byte*. Поглинає вхідні байти і обновлює кільцевий хеш на основі алгоритму Рабіна. Результат *true* або *false*, повідомляє чи потрібно розбивати потік на блок в даний момент чи продовжувати поглинання.

Клас *Stream* представляє потік даних, отриманих зі списку файлів наданих користувачем. Відповідає за утворення блоків даних для дедублікації. Атрибути та методи:

- *files*, атрибут типу *List[String]* – список файлів на основі яких будувати потік даних.
- *rabinPoly*, атрибут типу *RabinPoly* – містить об’єкт класу *RabinPoly*.
- *nextBlock*, метод з результуючим типом *Block* – поступово вичитує дані з файлу викликає метод *consume* на атрибуті *rabinPoly* для кожного вхідного байту. Коли *consume* сповістить про момент розбиття, утворює об’єкт класу *Block* що і є результатом методу.

Клас *Block* представляє блок даних для дедублікації отриманих з розбиття вихідного потоку даних. Відповідає за стиснення, шифрування та обчислення хеш значення даних. Атрибути та методи:

- *data*, атрибут типу *bytes* – масив байтів даних.
- *size*, атрибут типу *int* – довжина масиву байтів оригінальних даних.
- *hash*, атрибут типу *Hash* – хеш значення блоку.

- *compressed,* атрибут типу *boolean* – вказує чи масив байтів даних стиснутий.
- *encrypted,* атрибут типу *boolean* – вказує чи масив байтів даних зашифрований.
- *computeHash,* метод з результуючим типом *Hash* – обчислює хеш значення блоку даних використовуючи хеш-функції, результат зберігається в атрибуті *hash.*
- *compress,* метод – проводить стиснення даних, результат зберігається в атрибуті *data,* якщо змінено розмір блоку то новий записується в атрибуті *size.*
- *encrypt,* метод – проводить шифрування даних, результат зберігається в атрибуті *data,* якщо змінено розмір блоку то новий записується в атрибуті *size.*

Клас *Hash* представляє хеш значення отримане на основі блоку даних. Відповідає за представлення хешу у 16-ковій системі числення та порівняння з іншими хешами. Атрибути та методи:

- *data,* атрибут типу *bytes* – масив байтів який містить дані хеш значення.
- *equals,* метод вхідним параметром якого є інших хеш типу *Hash* – порівнює вхідне хеш значення із власним.
- *hex,* метод з результуючим типом *String* – відображає хеш значення у 16-ковій системі числення.

Клас *Coordinator* призначений для координування та розподілення блоків між робітниками у кластері. Атрибути та методи:

- *workers,* атрибут типу *List[Worker]* – список відомих робітників у вигляді об'єктів класу *Worker.*
- *distribute,* метод вхідним параметром якого є об'єкт типу *Hash* – на основі 16-вого числа отриманого від вхідного об'єкту хешу (викликаючи метод *hex)* вирішує якому робітнику делегувати завдання. Результатом виклику методу є об'єкт типу *Worker.*
- *finish,* метод вхідним параметром якого є ідентифікатор резервного копіювання, завершує процес дедублікації та формує об'єкт резервної копії класу *Backup.*

Клас *Worker* представляє робітника системи який відповідальний за обробку вхідних блоків даних та перевірки хешів на існуючість у системі. Атрибути та методи:

- *id,* атрибут типу *int* – унікальний ідентифікатор робітника у кластері.
- *index,* атрибут типу *Index* – містить об'єкт класу *Index.*
- *storage,* атрибут типу *Storage* – містить об'єкт класу *Storage.*
- *testHash,* метод вхідним параметром якого є об'єкт класу *Hash* – перевіряє чи вхідний хеш існує в системі використовуючи атрибут *index* та викликаючи відповідний метод.
- *append,* метод вхідним параметром якого є об'єкт класу *Block* – зберігає блок даних в хмарних сховищах використовуючи атрибут *storage* та викликаючи на ньому відповідний метод.

Клас *Index* представляє індекс систему у вигляді хеш таблиці яка зберігає усі відомі хеші системі у вигляді ключа та посилання на блок даних у хмарі у вигляді значення. Атрибути та методи:

- *path,* атрибут типу *String* – містить шлях до індексу на диску.
- *size,* атрибут типу *long* – розмір індексу байтах.
- *table,* атрибут типу *bytes* – масив байтів що відповідає структурі хеш таблиці.
- *put,* метод вхідними параметрами якого є хеш типу *Hash* та шлях до блоку у хмарі типу *String* – добавляє в хеш-таблицю нову пару ключ-значення, де ключом є вхідний хеш, а значенням шлях до блоку даних у хмарі.
- *testHash,* метод вхідним параметром якого є хеш типу *Hash* – перевіряє чи вхідний хеш зберігається в хеш-таблиці.

Клас *Storage* представляє хмарні сховища. Відповідає за збереження блоків даних та резервних копій. Атрибути та методи:

- *provider,* атрибут типу *String* – містить інформацію про постачальника хмарних сховищ.
- *saveBlock,* метод вхідним параметром якого є блок даних типу *Block* – зберігає вхідний блок даних у хмарних сховищах.
- *saveBackup,* метод вхідним параметром якого є резервна копія типу *Backup* – зберігає резервну копію даних у хмарних сховищах.

З розглянутих видів дедублікації обрано дедублікацію на рівні блоків. Не зважаючи на те, що даний вид дедублікації є складнішим у реалізації та проектуванні, дедублікація на рівні блоків забезпечує зменшення вхідного об'єму даних до 95% [1]. В той час як дедублікація на рівні екземплярів напряму залежить від типу даних резервних копій, і найменші зміни у вихідних файлів вимагатимуть повторного запису їх загалом. Також, дедублікація на рівні блоків, забезпечує рівномірне розподілення задач та їх об'єму між робітниками у кластерному середовищі, що сприяє максимальному використанні наявних обчислювальних ресурсів, та рівномірному розподіленню даних між окремими вузлами хмарного сховища. Раніше розглянуто та детально описано чотири типи дедублікації: вихідна, цільова, потокова та гібридна. З поміж них необхідно вибрати той тип дедублікації який найкраще підходить для розподілених обчислень і збереження даних у хмарних сховищах, щоб задовільними цілі дослідження.

Вихідна дедублікація виконується на пристрої, де знаходяться вихідні дані. Оскільки вихідна дедублікація працює на комп’ютері користувачів, то єдиним способом збільшити її швидкодію є вертикальна масштабованість, шляхом збільшення ресурсів комп’ютера користувача (покупка нового процесора, оперативної пам’яті та дискового простору). Проте, на комп’ютері користувача містять і інакші програми його повсякденного користування, а диск використовується для зберігання оригінальних даних. Тому, виділити додаткове місце для резервних копій буде фінансово витратно. Враховуючи ці фактори, вихідна дедублікація не підходить для цілей дослідження та завдань розроблюваної інтелектуальної системи. Цільова дедублікація виконується на цільовому центральному сервері, де на кінцевому етапі будуть зберігатись дані. Даний підхід краще підходить, ніж попередній тим, що підключає роботу центрального серверу, тим самим акумулюючи дані з різних джерел дедублікації. Також такий підхід підходить краще для масштабованості системи, оскільки обчислювальні ресурси, які встановлюють на виділені сервери зазвичай більшої продуктивності; а також, можна розраховувати на максимальне використання ресурсів, оскільки сервер дедублікації буде містити лише програмне забезпечення яке стосується лише дедублікації та збереження даних. Отже, даний тип дедублікації вигідніший з точки зору масштабованості. Проте, враховуючи той факт, що цільова дедублікація вимагає повного пересилання даних, збереження на диск і тільки тоді проводити відсічення дублікатів, оптимізація обмежується лише горизонтальною масштабованістю. Вертикальна масштабованість відпадає, оскільки не зважаючи на кількість процесорів, всі дані проходять через диск, який по своїй природі однопоточний і повільніший ніж оперативна пам'ять.

Потокова дедублікація по своїм природі створена для вирішення проблеми з диском у цільовій дедублікації. Процес розбиття потоку даних відбувається на момент отримання даних на сервері та відсічення дублікатів перед записом на диск. Такий підхід збільшує навантаження на оперативну пам'ять, тримаючи в пам’яті певне вікно даних, де відбувається розбиття і відсічення, що мінімізує навантаження на диск. Також це дозволяє масштабувати систему вертикально. Проте, все ще є єдиний недолік такого підходу, а саме навантаження на мережу та інтернет трафік користувача, оскільки всі дані користувача для резервного копіювання необхідно надіслати до серверу. Це стає великою проблемою коли виконується періодичне резервне копіювання і відсоток змін між копіями дуже малий.

Гібридна дедублікація – це суміш вихідного, цільового та потокового типу дедублікації. А саме розбиття даних на блоки та відсічення дублікатів відбувається на стороні користувача, а на сервер надсилаються лише нові блоки даних. Варто підкреслити, що комп’ютер користувача напряму не втягнений у процес усунення дублікатів напряму, програма-клієнт користувача надсилає список хеш значень блоків, а сервер вказує на ті хеш значення, які йому не відомі. Тобто робота з індексом лежить повністю на сервері. Таким чином можна уникнути додаткової навантаженості на комп’ютер користувача, на мережу та інтернет трафік. Також, враховуючи природну випадковість хеш значень, роботу з блоками легко розподілити між вузлами кластеру, тим самим масштабуючи систему щоб досягти максимальної продуктивності серверного обладнання. Враховуючи ці фактори, було обрано гібридний тип дедублікації для розробки інтелектуальної системи. Розбиття вхідного потоку даних на блоки для дедублікації на рівні блоків передбачає два алгоритми - з фіксованою довжиною та зміною. Згідно підрозділу 1.2 підхід з фіксованою довжиною передбачає розбиття потоку на рівні частини попередньо вибраної довжини, в той час як в підході зі змінною довжиною, точку розбиття вибирає алгоритм, на основі даних самого потоку. Алгоритм при розбитті зі змінною довжиною не гарантує того що блоки будуть мати однакову довжину, проте можливо встановити мінімальний і максимальний розмір блоків. Для прикладу візьмемо потік даних де одиницею є літера української абетки, а середня довжина блоку 5 літер. Вибравши літери від *А* до *Ф* отримаємо 5 блоків даних. Проте, якщо вихідний потік зазнає змін, наприклад цифру *1* буде виставлено між буквами *Ґ* та *Д*, алгоритми поведуть себе по різному. На рис. 6а подано заданий приклад, використовуючи підхід з розбиттям з фіксованою довжиною. Перший блок не зазнав змін, а отже вважається існуючим і повторне копіювання непотрібне. Проте усі наступні блоки будуть вважатись для системи новими, оскільки їхній контент буде відрізнятись від оригінальних.

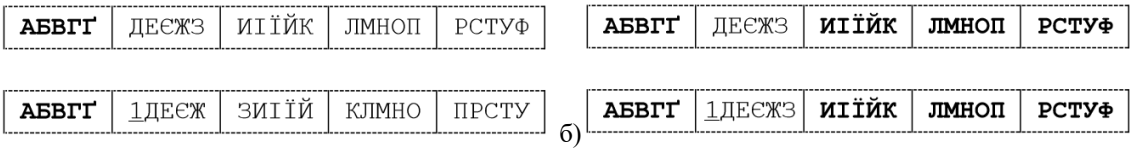

Рисунок 6 – Розбиття а) з фіксованою довжиною та б) зі змінною довжиною

На рис. 6б подано заданий приклад, використовуючи підхід з розбиттям зі змінною довжиною. На відміну від розбиття з фіксованою довжиною, алгоритм зі змінною довжиною, залишив літеру *З* у другому блоці, хоча це і збільшило вихідний розмір блоку, проте усі наступні блоки залишились ідентичними

до оригінальних. Таким чином, змінився лише той блок, в регіоні якого відбувся зсув. Алгоритм розбиття зі змінною довжиною працює не лише зі зсувом, а також із будь-якими змінами, використовуючи алгоритм кільцевого хешу. В якості алгоритму розбиття потоку даних на блоку було вибрано цифровий підпис Рабіна, який базується на основі обчислення кільцевого хешу. Найоптимальнішим розміром вікна є масив довжиною 16 біт, оскільки розміри більших значень не показують значного покращення роботи алгоритму [16]. А отже на кожен вхідний байт даних буде обчислено кільцеве хеш значення, для якого буде застосовано побітову операції I (AND) зі значенням середньої довжини блоку. Для даного дослідження було обрано середню довжину блоку 4096, мінімальну 2048, а максимальну 8192. Згідно досліджень вказаних вище, саме такі параметри дозволяють отримати максимальну точність розбиття даних при вікні довжиною 16 біт. Отже, виконуючи кожен раз для вхідного байту побітову операції AND над кільцевим хешем та середнім значенням довжини блоку – *A*, ми отримаємо число в межі від нуля до *A*. Якщо отриманий результат буде дорівнювати заданому шаблону, то поточний байт буде вважатись точкою розбиття потоку на наступний блок. Враховуючи межі довжини блоку, якщо точку знайдено раніше ніж мінімальна довжина, то вона ігнорується до моменту виходу за мінімальну довжину. Якщо точку розбиття не знайдено, а довжина блоку досягає максимальної довжини, то розбиття відбувається в момент досягнення максимальної довжини блоку. Такий випадок можливий при заповнені потоку даних повторюваними байтами, наприклад деякий регіон даних заповнений нулями, а отже кільцевий хеш, буде відмінний від заданого шаблону в будь-якій точці проблемного регіону.

Найважчим, з точки зору обчислень, в інтелектуальній системі є наповнення індексу та пошук в ньому ідентичних блоків даних на основі їхніх хеш значень. А отже дане завдання необхідно масштабувати горизонтально між вузлами кластеру системи та горизонтально між процесорами робочих машин. Для горизонтального масштабування необхідно вибрати ключ розподілу. Враховуючи випадкову природу хеш значень блоків, зобразивши хеш значення у вигляді чисел N-ної системи числення, можна майже задарма отримати зважений ключ розподілу. Для прикладу візьмемо 16-кову систему числення (оскільки даний тип найпоширеніший у представленні хеш значень). Взявши перше число хеш значення, ми отримає число від 1 до 16, зі ймовірністю появи . Таким чином ми отримаємо рівномірний розподіл блоків даних між 16-ма вузлами системи. Якщо в системі наявно менше число вузлів то вибір номеру вузла є остача від ділення першого числа хеш значення на кількість вузлів системи. Якщо вузлів є більше ніж 16, то береться друге число, тим самим отримуючи 256 комбінацій (16^2).

Визначившись з горизонтальним розподілом блоків даних між робітниками, можна досягти паралельного запису блоків у сховище кожного вузла. Але також, таким чином можна досягти і реалізації розподіленого індексу. Пошук дублікатів, використовуючи індекс, є лише процесорно-залежним завданням, оскільки індекс зберігається в оперативній пам'яті тим самим уникаючи роботи з диском. А процесорно-залежні завдання можливо масштабувати вертикально, розпаралелюючи роботу між процесорами комп’ютера, що у нашому випадку – пошук хеш значень у індексі. Досліджувана інтелектуальна система дедублікації та розподілу даних у хмарних сховищах передбачає роботу з блоками байтами даних та контролює їх розміщення на основі індексу (хеш-таблиці) в хмарних сховищах. А отже структуру традиційної реляційної бази даних необхідно будувати лише для мета інформацій, як от перелік користувачів та їх резервних копій. На рис. 7 подвно схему реляційної бази даних, а саме дві таблиці: *Backup* та *User*, які з’єднані зв’язком багато до одного. Загалом цих даних достатньо для функціонування досліджуваної системи, оскільки інформацію про інші дані система зберігає у хмарних сховища і вони не підходять для реляційних відношень. Таблиця *User* містить дані про користувачів системи, її колонки:

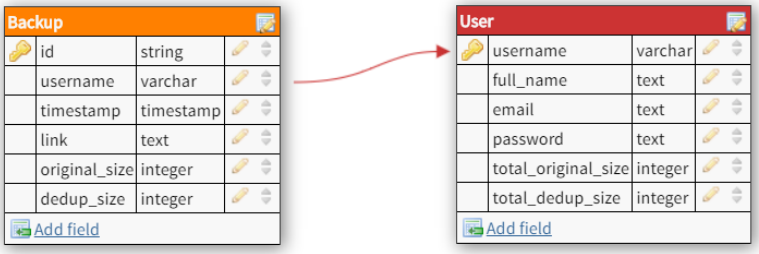

Рисунок 7 – Схема реляційної бази даних

- *username* – типу *varchar*, унікальний ключ таблиці, містить унікальне ім’я користувача і являється його ідентифікатором, може використовуватись для автентифікації користувача;

- *full_name* – типу *text*, містить повне ім’я користувача, використовується для формального звернення до користувача;
- *email* – типу *text,* електронна пошта користувача, може використовуватись для автентифікації користувача;
- *password* – типу *text,* хеш значення паролю, використовується для перевірки реального паролю під час автентифікації користувача;
- *total_original_size* – типу *integer,* загальний оригінальний розмір даних збережених користувачем, змінюється в залежності від добавлення нових резервний копій або їх видалення;
- *total_dedup_size* – типу *integer,* загальний розмір даних збережених користувачем після дедублікації, змінюється в залежності від добавлення нових резервний копій або їх видалення;

Маючи дані з колонок *total_original_size* та *total_dedup_size*, можна визначити коефіцієнт дедублікації: *1 - total_dedup_size / total_original_size,* та кількість місця на диску зекономлених завдяки дедублікації: *total_original_size - total_dedup_size*.

Таблиця *Backup* містить дані про резервні копії системи, її колонки:

- *id* – типу *text*, назва резервної копії, переважно складається з ідентифікатора користувача та з мітки часу створення резервної копії;
- *username* – типу *varchar*, унікальний ідентифікатор користувача, використовується для з’єднання з таблицею *User*, реалізовуючи зв'язок багато до одного;
- *timestamp* – типу *timestamp*, мітка часу створення резервної копії, використовується для визначення віку резервної копії;
- *link* – типу *text,* посилання на резервну копію у хмарному сховищі;
- *original_size* – типу *integer,* оригінальний розмір резервної копії;
- *dedup_size* – типу *integer,* розмір резервної копії після дедублікації;

Для того щоб дані в колонках *total_original_size* та *total_dedup_size* таблички *User* були завжди актуальними, необхідно створити тригер бази даних на табличці *Backup* на діях добавлення і видалення рядків. На рис. 8 подано sql тригер *user_total_change* зміни значень колонок *total_original_size* та *total_dedup_size*.

```
CREATE OR REPLACE TRIGGER user_total_change
    AFTER INSERT OR DELETE ON backup FOR EACH ROW
BEGIN
    IF TG_OP = 'INSERT' THEN
        UPDATE user set
            total_original_size = total_original_size + :new.original_size,
            total_dedup_size = total_dedup_size + :new.dedup_size
        WHERE username = :new.username;
    ELSIF TG_OP = 'DELETE' THEN
        UPDATE user set
            total_original_size = total_original_size - :old.original_size,
            total_dedup_size = total_dedup_size - :old.dedup_size;
        WHERE username = :new.username;
    END IF;
END;
```

Рисунок 8 – SQL тригер зміни колонок total_original_size та total_dedup_size

Даний тригер спрацює після успішного добавлення або видалення записів з таблиці *Backup*, який працює наступним чином: якщо буде добавлено новий рядок, то буде виконано операцію *update*, яка збільшить значення в колонках *total_original_size ma total_dedup_size* на *original_size* та *dedup_size* відповідно для зв’язаного користувача; якщо буде видалено старий рядок, то буде виконано операцію *update,* яка відповідно зменшить значення *total_original_size ma total_dedup_size.*

Для зберігання даних на чи то на дисковий простір, чи у хмарні сховища, необхідно використати власний формат файлів, оскільки зберігання блоків даних у реляційних базах буде надлишковими витратами на читання, запис та їхній індекс. Оскільки реляційні база даних створені для зберігання різних типів даних які мають невизначену схему, що створюється під час виконання програм. Проте у випадку з дедублікацією, можна оминути багато таких обмежень, оскільки уся структура і схема даних відома наперед. Отже, для зберігання блоку даних на диску, необхідно мати наступні метадані – порядковий номер, фактичний розмір блоку, оригінальний розмір блоку, хеш значення, алгоритм стиснення, алгоритм шифрування. Варто добавити, що фактичний розмір блоку може відрізнятись від оригінального, оскільки алгоритми стиснення та шифрування можуть змінити його. Порядковий номер блоку необхідний для його швидкого доступу під час відновлення резервних копій. На рис. 9 подано структуру файлу, який буде зберігати блоки даних.

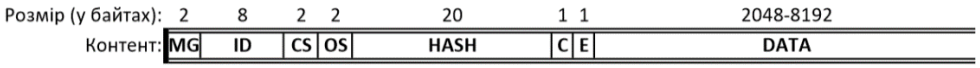

Рисунок 9 – Структура файлу зі списком блоків даних

На стороні користувача необхідно мати операційну систему *Windows* з встановленим *Microsoft C++ Runtime Library* 2015 і вище або операційну систему Linux. Також не залежно від операційної системи повинен бути встановлений браузер Chrome або його аналог Chromium. На стороні серверу, необхідне встановлення *JVM* (Java Virtual Machine). А також відкриті порти для доступу по *HTTP* протоколу. Також вимагається встановлення на основному вузлі кластеру одної із реляційних баз даних, таких як *Postgres* або *MySQL*. Для підключення хмарних сховищ необхідна оплачена та відкрита підписна на *Amazon S3*. В разі використання власного середовища для хмарних сховищ, необхідно мати сервер із операційною системою *Linux* та встановленим одним із безплатних *S3* середовищ – *LeoFS, RiakCS, Minio, Ceph, SwiftStack* та ін. Користувацький інтерфейс написаний на *JavaScript, Html* та *Css.* Програма-клієнт написаний на *Rust*. Програма-сервер на *Scala.* Інтелектуальна система дедублікації та розподілу даних у хмарних сховищах використовується в програмному забезпеченні для резервного копіювання даних. Дана система може використовуватись як окремий продукт, так і у вигляді модуля інакшої системи.

Інтелектуальна система дедублікації та розподілу даних у хмарних сховищах є багатомодульної та складається з наступних модулів: користувацького інтерфейсу, програми-клієнта, індексу та програми-сервера. Вхідні дані користувача це набір файлів для резервного копіювання. Програма-клієнт формує на основі цих даних потік байтів даних і розбиває його на блоки використовуючи алгоритм розбиття на основі кільцевого хешу. Для кожного отриманого блоку даних обчислюється хеш значення і перевіряється на наявність цього хешу в системі. Якщо таких хеш уже відомий то використовується посилання на цей хеш, замість усього блоку. Якщо ні, то даний блок зберігається в системі а хеш записується в індекс з посиланням на блок даних. Даний процес повторюється для кожного блоку до кінця потоку. В результаті вхідний потік зберігається як список із хешів для майбутнього відновлення даних з резервної копії. Розподіл даних між робітниками в кластері відбувається на основі хеш значення. Хеш значення представлене у вигляді набору чисел з шістнадцяткової системи числення. Для вибору необхідного робітника використовуються перше число хешу, в залежності від кількості робітників (це повинно бути обов'язково число степені двійки). Наприклад, якщо число робітників 16, то значення числа хешу дорівнює номеру робітника, якщо менше ніж 16, то число масштабується до необхідної основи. Якщо кількість робітників більше 16 то використовується наступне число робітників і т.д. Варто також добавити що індекс є розподіленим, тобто кожен робітник має свій власний індекс.

Модуль користувацького інтерфейсу написаний на мові програмування JavaScript. Для реалізації користувацького інтерфейсу як настільного додатку, використано бібліотеку ElectronJS, яка використовує потужності веб-браузера Chrome для відображення веб-сторінок поза межами браузера. Для побудови структури користувацького інтерфейсу було використано архітектурний шаблон модель-вигляд-контролер та його реалізацію AngularJS. Структурно модуль складається із:

- package.json – файл налаштувань, містить перелік підключених бібліотек та список задач побудови проекту.
- main.js – файл бізнес логіки головного вікна, містить конфігурації відкриття програми та перевірки локального кешу. В залежності чи користувач ідентифікований відкриває для нього основне вікно або вікно входу в систему.
- main.css – файл який містить таблиці стилів, описує зовнішній вигляд користувацького інтерфейу.
- main.html – файл опису структури головного вікна, його макет.
- login.html – файл вигляду вікна входу.
- dashboard.html – файл вигляду загальної інформації про резервні копії користувача, дозволяє створювати нові та відновлювати дані з існуючих.
- bridge.js – файл бізнес логіки для зв'язку з програмою-клієнтом.

Модуль програма-клієнт написаний на мові програмування *Rust.* Програма реалізована як архітектурний шаблон інтерфейс командного рядка. Тобто дана програма може використовуватись без користувацького інтерфейсу шляхом виклику першої з передачею вхідних аргументів. Проте даний підхід використовується лише для відлагодження. Для обробки командного рядки використовується бібліотека *Clap-rs*. Використовує *Reqwest* в якості HTTP клієнта. Для розбиття потоку на блоки використовує бібліотеку *Rabinpoly*, а для обчислення їх хешів – *SHA-1*. Структурно модуль складається:

- Cargo.toml – файл налаштувань, містить перелік підключених бібліотек та список задач побудови проекту.

- main.rs – точка входу в програму, містить обробку вхідний аргументів та виклик необхідних функцій.
- rabinpoly.rs – розбиття потоку даних на блоки зі змінною довжиною використовуючи алгоритм Рабіна.
- hash.rs – обрахунок хеш значень блоків дедублікації.
- http.rs – інтерфейс для виклику HTTP запитів.
- test_hashes.rs – перевірка хешів на існуючість.
- send_data.rs – надсилання нових блоків до сервера.
- backup.rs – композиція наявних функції та інтерфейсів для процесу резервного копіювання використовуючи дедублікації.
- recover.rs – відновлення даних з резервних копій.

Модуль програма-сервер написаний на мові програмування *Scala* та реалізує модель акторів за допомогою бібліотеки *Akka*. В якості HTTP серверу використовує *Akka-http*. Для робою з хмарними сховищами використовує бібліотеку *Aws-sdk*, яка надає інтерфейс для спілкування з *Amazon S3*. Структурно модуль складається із:

- build.sbt – файл налаштувань, містить перелік підключених бібліотек та список задач побудови проекту.
- MasterMain.scala – точка входу в систему головного вузла кластеру, координатора.
- WorkerMain.scala – точка входу в систему робітника, вузла кластеру.
- HttpServer.scala – файл налаштувань HTTP сервера та маршрутизації веб запитів.
- MasterActor.scala – актор-координатор, координує вхідні хеші і на основі їх значень вибирає робітника для делегування задач.
- WorkerActor.scala – актор-робітник, спілкується з індексом та виконує надсилання блоків дедублікації до хмарних сховищ.

Програма-сервер та її робітники використовують хмарні сховища для зберігання блоків даних та дерево об’єктів у вигляді резервних копій. В якості хмарного сховища необхідно використовувати сервіси сумісні з S3 інтерфейсом.

Клієнтську частину інтелектуальної системи призначено для використання на персональних комп’ютерах, які працюють на Linux, MacOS та Windows. Системні вимоги:

- Оперативна пам'ять 1 ГБ.
- Місце на диску - не менше ніж 256 МБ.
- Відео пам'ять – не менше ніж 256 МБ.

Серверну частину інтелектуальної системи призначено для використання у розподіленому кластерно середовищі, з щонайменше одним робочим вузлом який працює на Linux, MacOS та Windows. Системні вимоги для кожного вузла окремо:

- Оперативна пам'ять 4-16 ГБ.
- Місце на диску 1 ТБ.

Користувацький інтерфейс та програму-клієнт необхідно спочатку встановити на комп’ютері користувача. Користувацький інтерфейс викликається шляхом запуску новоствореної іконки на робочому столі або запуском за допомогою скрипта розташованого в директорії встановленої системи. Програму-клієнт не потрібно додатково викликати, це відбувається автоматично в залежності від маніпуляцій з користувацьким інтерфейсом.

Для успішного запуску програми-сервера та індексу необхідно спочатку налаштувати робоче розподілене середовище у вигляді кластеру. Необхідно знати усі мережі IP-адреси кожного вузла кластеру і вказати їх під час запуску програми-сервера. Індекс будується автоматично при старті кожного робітника програми-клієнта.

Точкою входу зі сторони користувача є користувацький інтерфейс у вигляді настільного додатку. Точкою входу зі сторони сервера є скрипт запуску кластерного середовища.

Вхідними даними для користувацького інтерфейсу є маніпуляції користувача, список файлів для дедублікації та посилання на резервну копію для відновлення даних. Вхідними даними для програми-клієнта є файли для дедублікації, список блоків дедублікації для надсилання. Вхідними даними для програми-сервера є назва резервної копії, список хешів вхідних блоків дедублікації та унікальні блоки дедублікації. Вхідними даними для індексу є хеші та посилання на блоки у сховищі.

Вихідними даними для користувацького інтерфейсу є статус процесу резервного копіювання, посилання на резервну копію, файли резервної копії. Вихідними даними для програми-клієнта є усі блоки дедублікації, список хешів блоків дедублікації. Вихідними даними для програми-серверу є резервна копія. Вихідними даними для індексу є відповідь на наявність хешу блоку в системі.

Після розгортання серверу, користувач має змогу підключитись до нього за допомогою програми-клієнта. Проте спочатку необхідно автентифікуватись у системі. В разі неуспішної автентифікації (неправильно введена поштова скринька або пароль), програма попросить користувача, перевірити дані та ввести їх знову. Після введення правильного паролю, та натиску кнопки *Увійти,* система успішно автентифікує користувача та головне вікно користувацького інтерфейсу (рис. 10). Головне вікно програми складається з двох частин – бічної панелі та головного поля. На бічній панелі є можливість вибрати зміст активного поля – резервні копії або налаштування. За замовчуванням, при вході користувача в систему, йому відкривається поле із резервними копіями, на якому зображений список із раніше створених резервних копій. Інтерфейс дозволяє проводити сортування та пошук резервних копій за змістом рядків. Також присутня кнопка *Добавити*, яка дозволяє створити нову резервну копію, при натисканні на яку, відкривається вікно вибору файлів для резервного копіювання.

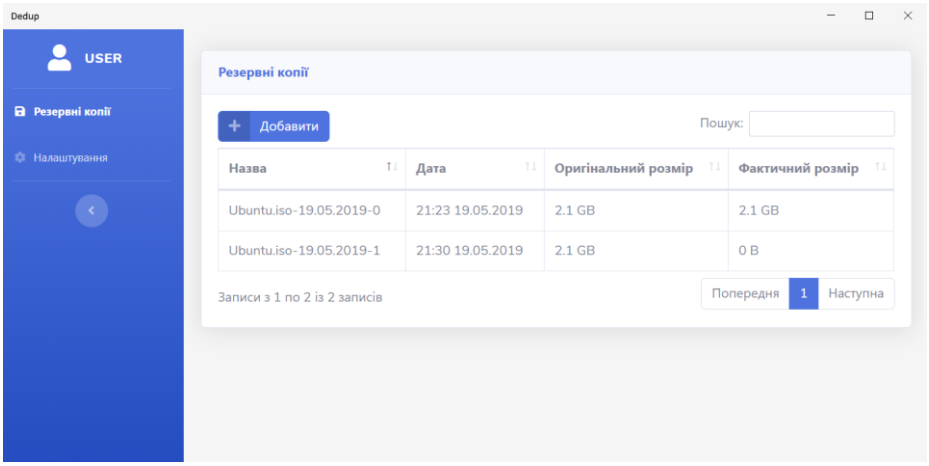

Рисунок 10 – Головне вікно програми

Для тестового запуску, вибрано документ *Диплом.docx,* розміром 5 мб. Після вибору файлу та натискання кнопки *Вибір* розпочнеться процес дедублікації, а саме: створення потоку даних на основі цього файлі та розбиття його на блоки; обчислення хеш значення кожного блоку та перевірка на існуючість його в системі; надсилання нових блоків даних на сервер; збереження резервної копії на хмарному сховищі та відображення її у програмі користувача (рис. 11-12). Для тестування процесу дедублікації під час резервного копіювання, необхідно добавити файл *Документ* ще раз і подивитись на зміни поля *Фактичний розмір.* На рис. 13 подано вікно з резервними копіями де було вдруге добавлено файл *Документ.* Як видно з рисунку, назва резервної копії є *Документ.docx-19.05.2019-1* а її фактичний розмір – 0 байтів, тобто на сервері не було збережено жодних нових блоків. На рис. 14 подано останні повідомлення кластеру для резервної копії *Документ.docx-19.05.2019-1.* Оскільки усі дані уже відомі системі, в журналі немає повідомлень щодо нових блоків, а отже є лише два сповіщення, про початок та кінець. Останнє повідомлення, також вказує що система отримала 0 нових байтів та 0 нових блоків.

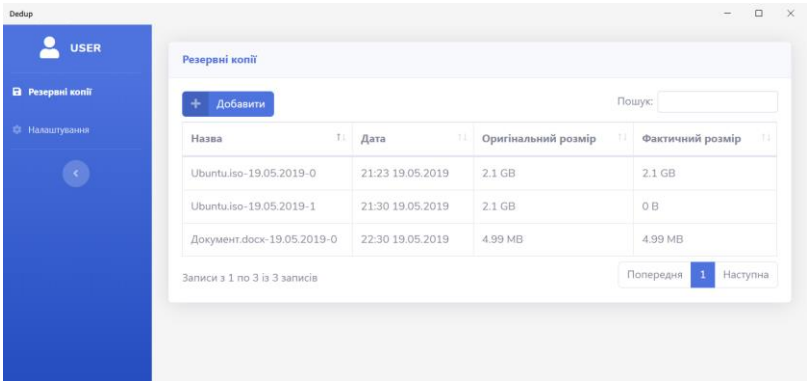

Рисунок 11 – Вікно резервних копій із доданим документом

```
22:29:570 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 0cacd9ccc3158f2e13c43d881d323289c2d61e94 : 4348
22:29:577 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 496a96d3bd7a0f7ffccfff98bb095d7d65617741 : 7379
22:29:580 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 62c63821f4ad60cd9ff39d2f902160183e651f7b : 5256
22:29:581 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 273094160eb75ab9e6641269a2b632297e606cc3 : 4269
22:29:589 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : c015ee1685fb0e18bbaa67573046a6029a251a24 : 7275
22:29:597 [worker-2] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : db7c7cbfbc805990074866e19d49d969565ee8f5 : 3679
22:29:598 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 6abb04c8a488900fefe4e0290b676e482aa319ce : 5352
22:29:607 [worker-0] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 726b41f26b99659422d191ac17239c50465a42e3 : 6479
22:29:615 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 655bf7aabc4efeb02111d58b6bcb79a895e98f7a : 3359
22:29:620 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 6b6bc85ccadb6512801a29d4dfe27d35cabcce3d : 2680
22:29:627 [worker-0] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : fe2507a3c018a6222597eb9bb382360af9dbeeb9 : 3742
22:29:633 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : ad3bae80b2634facd42c415022ed86995e68d7dc : 4516
22:29:639 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 0d167488d55a8352e2e3cb5f657190d2d1ba2e56 : 7184
22:29:643 [worker-2] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : dfb7c7920d70fdd1220f7e911d504812212f53cc : 2451
22:29:646 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 8142e05b10d6ae3e6a380de4c0e4d4deb508e596 : 3095
22:29:650 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 40d9cd5f2f181e2c5730bbc6e0798b6bad254d3e : 5281
22:29:654 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 0650cf5d3e9011dfa0f05045b8215cee4bd0d1f7 : 2078
22:29:659 [worker-2] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 15f30fba6da4780a91b976f6bf6d8695ea2634d9 : 3223
22:29:667 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : eb135e66faf6c9b13334b5cb3a65d77aff5538f2 : 6251
22:29:670 [worker-0] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 3b08598a0aa641844b4f0dd640ef8c79cfc5dc9e : 2154
22:29:671 [worker-2] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 98638d137342f1b12299c272f06a76bf608e73b3 : 3965
22:29:680 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 8d6aece9d8dc0a2ae56c45b89ef41134463111c6 : 7649
22:29:685 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 276cbfe9b2a705e80039c5a4099da0442ed6781f : 7759
22:29:690 [worker-2] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 1a76827c2321b5774429dc4d47b50713aa03b0f0 : 6946
22:29:694 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 04d4c76e47733da3b0aa9eecb2da299987733c1f : 4532
22:29:696 [worker-2] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 587983b6b8f6e98f79a4341fd960dfa99eea7c34 : 3422
22:29:697 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : a8e4ac9aa00c8d615748e5543b1cdff1a9e72455 : 7856
22:29:702 [worker-2] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : dc88178f2db20baab4ed6ec4860941a33255f4e4 : 7641
22:29:708 [worker-0] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : bb1c612876f77288afb55af6297315d54d1f2b5a : 2120
22:29:710 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 4168d41cb2435eeff3ad09bdf220d771873599e3 : 7627
22:29:719 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 4b63f065137cf98601cd790896c99e5983abf636 : 5349
22:29:723 [worker-1] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 45ab2cdd0f5b69741c9fd591da207b157930ac8d : 5016
22:29:733 [worker-2] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : dcb24f1d6880050e7981b68db7a72f00926fecfb : 5122
22:29:738 [worker-0] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 3b093efda2be455a94a0cc282e293844f3a044b1 : 6418
22:29:745 [worker-0] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 72078081259bd09985f3e13760f7a3cd56286bca : 3721
22:29:746 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : 6f62f16678eb3b8a6f613de66493df08fb6dd329 : 2704
22:29:755 [worker-0] INFO - NEW BLOCK : Документ.docx-19.05.2019-0 : f1b49172485754ff80780cc5da8cf58104f0907e : 2622
22:30:012 [coordinator] INFO - FINISHED : Документ.docx-19.05.2019-0 : ALL_BYTES=52232394 : NEW_BYTES=52232394 : BLOCKS=10435(new=10435)
```

Рисунок 12 – Журнал повідомлень для першої резервної копії

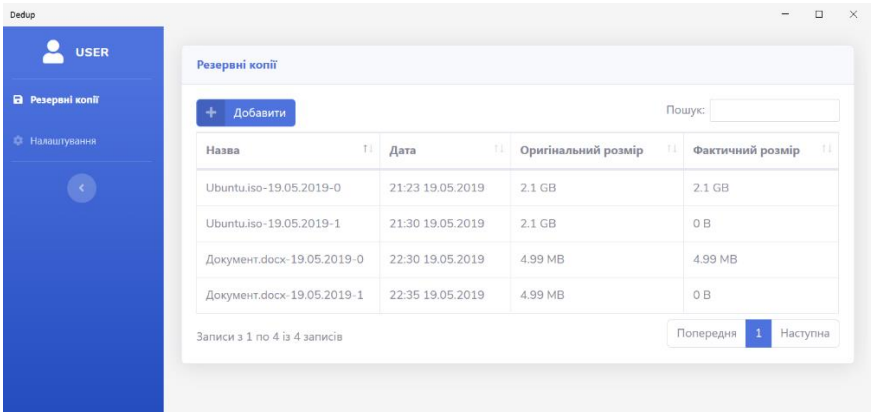

Рисунок 13 – Вікно резервних копій із вдруге доданим документом

```
22:33:703 [coordinator] INFO - STARTED : Документ.docx-19.05.2019-1 : USER=user
22:35:214 [coordinator] INFO - FINISHED : Документ.docx-19.05.2019-1 : ALL_BYTES=52232394 : NEW_BYTES=0 : BLOCKS=10435(new=0)
```

Рисунок 14 – Журнал повідомлень для другої резервної копії

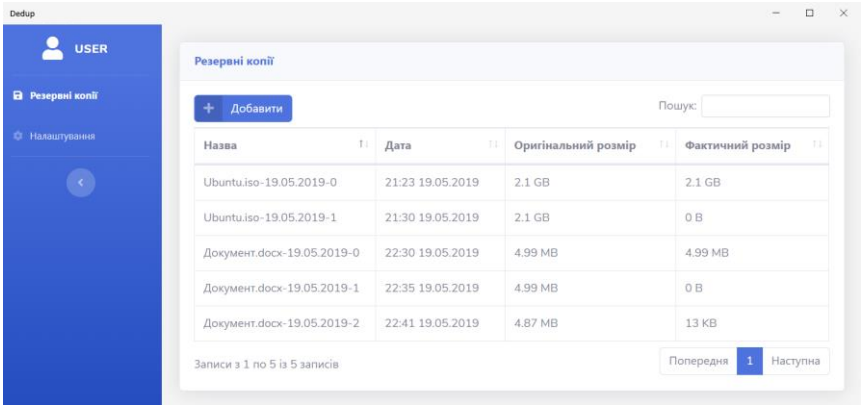

Рисунок 15 – Вікно резервних копій із втретє доданим документом

Для того щоб протестувати чи алгоритм розбиття потоку даних на блоки працює правильно, та будь-які не значні зміни в документі не потягнуть за собою заміну всіх блоків після точки змін. Для цьо-

го відкриємо оригінальний файл *Документ* та видалимо звідки трохи інформації і повторимо кроки для того щоб зробити нову резервну копію. На рис. 15 подано список резервних копій, з новоствореною *Документ.docx-19.05.2019-2,* яка містить файл *Документ* з дещо зміненими даними в середині файлу. Як видно з отриманих результатів, оригінальний розмір 4.87 мегабайт (оскільки було видалено частину даних), а фактичний – 13 кілобайт. Що означає, що алгоритм розбиття блоків виконав своє завдання, і змінив лише ті блоки, де відбувалися зміни. На рис. 16 подано останні повідомлення кластеру для резервної копії *Документ.docx-19.05.2019-2.* Згідно цих повідомлень, було добавлено лише 2 нові блоки (разом 13 кілобайт), які охоплюють те місце де було видалено дані з оригінального документу.

```
22:39:201 [coordinator] INFO - STARTED : Документ.docx-19.05.2019-2 : USER=user
22:40:365 [worker-3] INFO - NEW BLOCK : Документ.docx-19.05.2019-2 : 6019aef800db07e4c1f7b7a2a8236b235c0ff344 : 7651
22:40:387 [worker-0] INFO - NEW BLOCK : Документ.docx-19.05.2019-2 : f5eb3712d8ead6a4637dc82989acde77ffd64abf : 5246
22:41:856 [coordinator] INFO - FINISHED : Документ.docx-19.05.2019-2 : ALL_BYTES=5106565 : NEW_BYTES=12897 : BLOCKS=10415(new=2)
```

Рисунок 16 – Журнал повідомлень для другої резервної копії

Також була зменшена загальна кількість байтів та блоків.Отже, під час тестового запуску, файл *Документ* було тричі надіслано для резервного копіювання, що завдяки дедублікації, займає на сховищі даних лише 4.99 МБ + 131 КБ = 5.1 МБ. Тоді, якщо би копіювання відбувалось без дедублікації, то фактичний розмір був би 4.99 МБ + 4.99 МБ + 4.87 МБ = 14.85 МБ.

Висновки

1. Здійснено аналіз сучасних технологій проектування та створення систем у сфері резервного копіювання та відновлення даних, здійснено порівняльний аналіз технічних реалізацій та алгоритмів, застосованих в реалізаціях Загальною метою розроблення є створення інтелектуальної системи дедублікації та розподілу даних у хмарних сховищах, що дозволить використовувати її в програмному забезпеченні резервного копіювання даних для усунення дублікатів та підвищення пропускної здатності вхідного потоку даних та загального швидкості процесу. Досліджувана інтелектуальна система дедублікації та розподілу даних у хмарних сховищах призначена довершити існуючі процеси дедублікації та їх інтеграцію з хмарними технологіями. Місцем застосування розроблюваної системи є підприємства або персональні користувачі метою яких є резервне копіювання даних. Було розглянуто відомі системи резервного копіювання, використовуючи дедублікацію даних у хмарних сховищах, а також їхні плюси та мінуси. Згідно їх опису, дані системи показують високі коефіцієнти дедублікації, а також можливість інтеграції в існуючі програми резервного копіювання та підключення хмарних сховищ різних постачальників і навіть приватних сховищ власного виробництва (за умови використання інтерфейсу S3). Проте оглянуті системи використовують хмарні сховища лише як кінцеву частину збереження даних, в той час як досліджувана інтелектуальна система націлена на повне використання хмарних технологій шляхом розподілу даних та завдань дедублікації. Таким чином, в той час як оглянуті аналогічні системи працюють на односерверних монолітних машинах, досліджувана система дозволяє підключати комп’ютерну мережу у вигляді обчислювального кластеру для виконання завдань дедублікації. Ефектом від впровадження інтелектуальної системи дедублікації та розподілу даних у хмарних сховищах є:

- зменшення витрати на збереження резервних копій – процес дедублікації передбачає усунення дублюючих даних, що є критично важливим для періодичних резервних копій, де відсоток зміни даних між ними досить малий;
- збільшення швидкодії, використовуючи алгоритми розподілення даних між робочими вузлами кластеру системи;
- можливість роботи системи на віддалених хмарних серверах та збереження даних у хмарних сховищах;
- легка інтеграції у існуючі рішення резервного копіювання даних.

2. Побудовано засобами UML концептуальну модель інтелектуальної інформаційної системи дедублікації та розподілу даних у хмарних сховищах, задля автоматизації та ефективної роботи ‘програмного забезпечення у сфері резервного копіювання та відновлення данихю

3. Проаналізовано (наведено недоліки та переваги використання різних підходів) та обрані ефективні методи вирішення задач: гібридну дедублікацію на рівні блоків, розбиття потоку даних на основі цифрового відбитку Рабіна, розподіл даних на основі хеш значень блоків дедублікації та використання розподіленого індексу. В ході аналізу засобів вирішення задач було обрано мову програмування Rust для написання клієнтської частини, мову програмування Scala для серверної частини, інструментарій Akka для менеджменту розподілених обчислень та Amazon S3 в якості хмарного сховища. Розроблено інтелектуальну систему дедублікації та розподілу даних у хмарних сховищах, наведено опис програмного забезпечення, розглянуті кроки для роботи користувача.

4. Розроблено інтелектуальну систему дедублікації та розподілу даних у хмарних сховищах. Проведено тестування роботи проектованої системи та наведено декілька контрольних прикладів, проаналізо-

вано отримані результати. Підсумувавши вищесказане, можна зробити висновок, що мета розробки досягнута. Побудовано інтелектуальну систему дедублікації та розподілу даних у хмарних сховищах, яка при доробці систем перевірки цілісності резервних копій та оптимізації дискового простору після видалення старих резервних копій може використовуватись на практиці.

Список літератури

[1] Understanding Data Deduplication [Електронний ресурс]. – Режим доступу: https://www.druva.com/understanding-data-deduplication

[2] Explaining deduplication rates and single-instance storage to clients [Електронний ресурс]. – Режим доступу: https://searchitchannel.techtarget.com/tip/Explaining-deduplication-rates-and-single-instance-storage-to-clientshttp://zakon.rada.gov.ua/laws/show/2469-19

[3] Inline vs. post-processing deduplication appliances [Електронний ресурс]. – Режим доступу: https://searchdatabackup.techtarget.com/tip/Inline-vs-post-processing-deduplication-appliances

[4] Introduction to Data Deduplication [Електронний ресурс]. – Режим доступу: https://www.petri.com/data-deduplication-introduction

[5] Rabin M. O. Fingerprinting by random polynomials / M. O. Rabin // Center for Research in Computing Technology Harvard University Report – Harvard, 1981.

[6] Tanenbaum A.S. Distributed Systems / A.S. Tanenbaum, M. van Steen. – Upper Saddle River : Pearson Prentice Hall, 2017. – 15 с.

[7] Amdahl G. The validity of the single processor approach to achieving large-scale computing capabilities. / G. Amdahl. – Atlantic City : Proceedings of AFIPS, 1967.

[8] Using StorReduce for cloud-based data deduplication [Електронний ресурс]. – Режим доступу: https://cloud.google.com/solutions/partners/storreduce-cloud-deduplication

[9] OpenDedup Overview [Електронний ресурс]. – Режим доступу: https://opendedup.org/odd/overview/

[10] 10. Rumbaugh J. The unified modeling language reference manual / J. Rumbaugh, I. Jacobson, G. Booch // Addison Wesley Longman Inc. – 1999.

[11] Rolling hash, Rabin Karp, palindromes, rsync and others [Електронний ресурс]. – Режим доступу: https://www.infoarena.ro/blog/rolling-hash

[12] Vysotska V. Methods based on ontologies for information resources processing / V. Vysotska, L. Chyrun, V. Lytvyn. - LAP Lambert Academic Publishing, 2016.

[13] Vysotska V. Information technologies of gamification for training and recruitment / V. Vysotska, N. Shakhovska. - Saarbrücken, Germany: LAP LAMBERT Academic Publishing, 2018.

[14] Висоцька, В.А. Особливості проектування та впровадження систем електронної комерції. / В.А. Висоцька // Комп'ютерні науки та інформаційні технології, Вісник Національного університету "Львівська політехніка". – Львів 2008. – № 629. – С. 34-45.

[15] Vysotska V. Web resources processing based on ontologies / V. Vysotska, V. Lytvyn. - Saarbrücken, Germany: LAP LAMBERT Academic Publishing, 2018.

[16] Vysotska V. Tekhnolohiyi elektronnoyi komertsiyi ta Internet-marketynhu / V. Vysotska. - Saarbrücken, Germany: LAP LAMBERT Academic Publishing, 2018.

[17] Vysotska V. Internet systems design and development based on Web Mining and NLP / V. Vysotska. - Saarbrücken, Germany: LAP LAMBERT Academic Publishing, 2018.

[18] Vysotska V. Computer linguistics for online marketing in information technology: Monograph / V. Vysotska. - Saarbrücken, Germany: LAP LAMBERT Academic Publishing, 2018.

[19] Lytvyn V. A Method of Construction of Automated Basic Ontology / V. Lytvyn, V. Vysotska, W. Wojcik, D. Dosyn // 1st International Conference Computational Linguistics and Intelligent Systems, COLINS'2017. – 21 April 2017, Kharkiv. – P. 75-83.

[20] Lytvyn V. Intelligent System Structure for Web Resources Processing and Analysis / V. Lytvyn, V. Vysotska, L. Chyrun, A. Smolarz, O. Naum // 1st International Conference Computational Linguistics and Intelligent Systems, COLINS'2017. – 21 April 2017, Kharkiv. – P. 56-74.

[21] Berko A. Features of information resources processing in electronic content commerce / Andriy Berko, Victoria Vysotska, Lyubomyr Chyrun // Applied Computer Science. ACS journal. – Volume 10, Number 2. – Poland, 2014. – ISSN 2353-6977 (Online), ISSN 1895-3735 (Print). – PP. 5-19.

[22] Берко, А.Ю. Методи та засоби оцінювання ризиків безпеки інформації в системах електронної комерції / А.Ю. Берко, В.А. Висоцька, І.В. Рішняк // Інформаційні системи та мережі. Вісник Національного університету "Львівська політехніка". – Львів 2008. – № 610. – С.20-33.

[23] Vysotska Victoria. Web Content Processing Method for Electronic Business Systems / Victoria Vysotska, Lyubomyr Chyrun // International Journal of Computers & Technology. – Vol 12, No 2. – December 2013. – PP. 3211-3220.

[24] Висоцька В.А. Моделювання етапів життєвого циклу комерційного web-контенту / В.А. Висоцька, Л.Б Чирун, Л.В. Чирун // Інформаційні системи та мережі. Вісник Національного університету "Львівська політехніка". – Львів 2011. – № 715. – С. 69-87.
[25] Берко, А.Ю. Алгоритми опрацювання інформаційних ресурсів в системах електронної комерції / А.Ю. Берко, В.А. Висоцька, Л.В. Чирун // Комп'ютерні науки та інформаційні технології. Вісник Національного університету "Львівська політехніка". – Львів 2008. – № 616. – Стор.128-136.
[26] Vysotska Victoria. Commercial Web Content Lifecycle Model: conference proceedings, November 16-19, 2011 / Victoria Vysotska, Lubomyr Chyrun, Lilya Chyrun // The 6th International Scientific and Technical Conference "Computer Sciences and Information Technologies" (CSIT'2011) which will be held November 16-19, 2011 at Lviv Polytechnic National University (Lviv, Ukraine) – Lviv 2011. – P. 160-163.
[27] Берко А.Ю. Проектування навігаційного графу Web-сторінок бази даних систем електронної комерції. / А.Ю. Берко, В.А. Висоцька // Комп'ютерні науки та інформаційні технології, Вісник Національного університету "Львівська політехніка". – Львів 2009. – № 638. – С. 3-14.
[28] Берко А.Ю. Семантична інтеграція неповних та неточних даних / А.Ю. Берко, В.А. Висоцька, В.В. // Збірник наукових праць «Системи обробки інформації. Безпека та захист інформації в інформаційних системах», Випуск 7 (79). – Харків 2009. – С. 93-98.
[29] Берко, А.Ю. Моделі та методи проектування інформаційних систем електронної комерції / А.Ю. Берко, В.А. Висоцька // Автоматизовані системи управління та прилади автоматики. Науково-технічний журнал. – Харків 2007. – № 138. – С.55-66.
[30] Алєксєєва К.А. Управління Web-ресурсами за умов невизначеності / К.А. Алєксєєва, А.Ю. Берко, В.А. Висоцька // Журнал «Технологический аудит и резервы производства». – Vol 2, No 2(22) (2015). – Харків, 2015. – ISSN (print) 2226-3780, ISSN (on-line) 2312-8372. – С. 4-7.
[31] Vysotska V. Designing features of architecture for e-commerce systems / Victoria Vysotska, Lyubomyr Chyrun // MEST Journal (Management Education Science & Society Technologie). – Vol.2 No.1. – P. 57-70.
[32] Vysotska V. Set-theoretic models and unified methods of information resources processing in e-business systems / Victoria Vysotska, Lyubomyr Chyrun // Applied Computer Science. ACS journal. – Volume 10, Number 3. – Poland, 2014. – ISSN 2353-6977 (Online), ISSN 1895-3735 (Print). – P. 5-2.

Стаття надійшла: 22.09.2019.

References

[1] Understanding Data Deduplication [Electronic resource]. – Available: https://www.druva.com/understanding-data-deduplication
[2] Explaining deduplication rates and single-instance storage to clients [Electronic resource]. – Available: https://searchitchannel.techtarget.com/tip/Explaining-deduplication-rates-and-single-instance-storage-to-clientshttp://zakon.rada.gov.ua/laws/show/2469-19
[3] Inline vs. post-processing deduplication appliances [Electronic resource]. – Available: https://searchdatabackup.techtarget.com/tip/Inline-vs-post-processing-deduplication-appliances
[4] Introduction to Data Deduplication [Electronic resource]. – Available: https://www.petri.com/data-deduplication-introduction
[5] Rabin M. O. Fingerprinting by random polynomials / M. O. Rabin // Center for Research in Computing Technology Harvard University Report – Harvard, 1981.
[6] Tanenbaum A.S. Distributed Systems / A.S. Tanenbaum, M. van Steen. – Upper Saddle River : Pearson Prentice Hall, 2017. – 15 c.
[7] Amdahl G. The validity of the single processor approach to achieving large-scale computing capabilities. / G. Amdahl. – Atlantic City : Proceedings of AFIPS, 1967.
[8] Using StorReduce for cloud-based data deduplication [Electronic resource]. – Available: https://cloud.google.com/solutions/partners/storreduce-cloud-deduplication
[9] OpenDedup Overview [Electronic resource]. – Available: https://opendedup.org/odd/overview/
[10] Rumbaugh J. The unified modeling language reference manual / J. Rumbaugh, I. Jacobson, G. Booch // Addison Wesley Longman Inc. – 1999.
[11] Rolling hash, Rabin Karp, palindromes, rsync and others [Electronic resource]. – Available:https://www.infoarena.ro/blog/rolling-hash
[12] Vysotska V. Methods based on ontologies for information resources processing / V. Vysotska, L. Chyrun, V. Lytvyn. - LAP Lambert Academic Publishing, 2016.
[13] Vysotska V. Information technologies of gamification for training and recruitment / V. Vysotska, N. Shakhovska. - Saarbrücken, Germany: LAP LAMBERT Academic Publishing, 2018.

[14] Vysotska, V.A. Osoblyvosti proektuvannya ta vprovadzhennya system elektronnoyi komertsiyi. / V.A. Vysotska // Komp″yuterni nauky ta informatsiyni tekhnolohiyi, Visnyk Natsional′noho universytetu "L′vivs′ka politekhnika". – Lviv 2008. – № 629. – S. 34-45.
[15] Vysotska V. Web resources processing based on ontologies / V. Vysotska, V. Lytvyn. - Saarbrücken, Germany: LAP LAMBERT Academic Publishing, 2018.
[16] Vysotska V. Tekhnolohiyi elektronnoyi komertsiyi ta Internet-marketynhu / V. Vysotska. - Saarbrücken, Germany: LAP LAMBERT Academic Publishing, 2018.
[17] Vysotska V. Internet systems design and development based on Web Mining and NLP / V. Vysotska. - Saarbrücken, Germany: LAP LAMBERT Academic Publishing, 2018.
[18] Vysotska V. Computer linguistics for online marketing in information technology: Monograph / V. Vysotska. - Saarbrücken, Germany: LAP LAMBERT Academic Publishing, 2018.
[19] Lytvyn V. A Method of Construction of Automated Basic Ontology / V. Lytvyn, V. Vysotska, W. Wojcik, D. Dosyn // 1st International Conference Computational Linguistics and Intelligent Systems, COLINS'2017. – 21 April 2017, Kharkiv. – P. 75-83.
[20] Lytvyn V. Intelligent System Structure for Web Resources Processing and Analysis / V. Lytvyn, V. Vysotska, L. Chyrun, A. Smolarz, O. Naum // 1st International Conference Computational Linguistics and Intelligent Systems, COLINS'2017. – 21 April 2017, Kharkiv. – P. 56-74.
[21] Berko A. Features of information resources processing in electronic content commerce / Andriy Berko, Victoria Vysotska, Lyubomyr Chyrun // Applied Computer Science. ACS journal. – Volume 10, Number 2. – Poland, 2014. – ISSN 2353-6977 (Online), ISSN 1895-3735 (Print). – PP. 5-19.
[22] Berko, A.YU. Metody ta zasoby otsinyuvannya ryzykiv bezpeky informatsiyi v systemakh elektronnoyi komertsiyi / A.YU. Berko, V.A. Vysotska, I.V. Rishnyak // Informatsiyni systemy ta merezhi. Visnyk Natsional′-noho universytetu "L′vivs′ka politekhnika". – Lviv 2008. – № 610. – S.20-33.
[23] Vysotska Victoria. Web Content Processing Method for Electronic Business Systems / Victoria Vysotska, Lyubomyr Chyrun // International Journal of Computers & Technology. – Vol 12, No 2. – December 2013. – PP. 3211-3220.
[24] Vysotska V.A. Modelyuvannya etapiv zhyttyevoho tsyklu komertsiynoho web-kontentu / V.A. Vysotska, L.B Chyrun, L.V. Chyrun // Informatsiyni systemy ta merezhi. Visnyk Natsional′noho universytetu "L′vivs′ka politekhnika". – L′viv 2011. – № 715. – Stor. 69-87.
[25] Berko, A.YU. Alhorytmy opratsyuvannya informatsiynykh resursiv v systemakh elektronnoyi komertsiyi / A.YU. Berko, V.A. Vysotska, L.V. Chyrun // Komp″yuterni nauky ta informatsiyni tekhnolohiyi. Visnyk Natsio-nal′noho universytetu "L′vivs′ka politekhnika". – Lviv 2008. – № 616. – Stor.128-136.
[26] Vysotska Victoria. Commercial Web Content Lifecycle Model: conference proceedings, November 16-19, 2011 / Victoria Vysotska, Lubomyr Chyrun, Lilya Chyrun // The 6th International Scientific and Technical Conference "Computer Sciences and Information Technologies" (CSIT'2011) which will be held November 16-19, 2011 at Lviv Polytechnic National University (Lviv, Ukraine) – Lviv 2011. – P. 160-163.
[27] Berko A.YU. Proektuvannya navihatsiynoho hrafu Web-storinok bazy danykh system elektronnoyi komertsiyi. / A.YU. Berko, V.A. Vysotska // Komp″yuterni nauky ta informatsiyni tekhnolohiyi, Visnyk Natsiona-l′noho universytetu "L′vivs′ka politekhnika". – Lviv 2009. – № 638. – S. 3-14.
[28] Berko A.YU. Semantychna intehratsiya nepovnykh ta netochnykh danykh / A.YU. Berko, V.A. Vysotska, V.V. // Zbirnyk naukovykh prats′ «Systemy obrobky informatsiyi. Bezpeka ta zakhyst informatsiyi v informa-tsiynykh systemakh», Vypusk 7 (79). – Kharkiv 2009. – S. 93-98.
[29] Berko, A.YU. Modeli ta metody proektuvannya informatsiynykh system elektronnoyi komertsiyi / A.YU. Berko, V.A. Vysotska // Avtomatyzovani systemy upravlinnya ta prylady avtomatyky. Naukovo-tekhnichnyy zhurnal. – Kharkiv 2007. – № 138. – S.55-66.
[30] Alyeksyeyeva K.A. Upravlinnya Web-resursamy za umov nevyznachenosti / K.A. Alyeksyeyeva, A.YU. Ber-ko, V.A. Vysots′ka // Zhurnal «Tekhnolohycheskyy audyt y rezervy proyzvodstva». – Vol 2, No 2(22) (2015). – Kharkiv, 2015. – ISSN (print) 2226-3780, ISSN (on-line) 2312-8372. – S. 4-7.
[31] Vysotska V. Designing features of architecture for e-commerce systems / Victoria Vysotska, Lyubomyr Chyrun // MEST Journal (Management Education Science & Society Technologie). – Vol.2 No.1. – P. 57-70.
[32] Vysotska V. Set-theoretic models and unified methods of information resources processing in e-business systems / Victoria Vysotska, Lyubomyr Chyrun // Applied Computer Science. ACS journal. – Volume 10, Number 3. – Poland, 2014. – ISSN 2353-6977 (Online), ISSN 1895-3735 (Print). – P. 5-2.

Відомості про авторів

Русин Богдан Павлович – д.т.н., професор,завідувач відділу методів та систет дистанційного зондування Фізико-механічного інституту НАН України, вул. Наукова, 5, м. Львів, 79601, Україна.

Погрелюк Любомир Володимирович – аспірант відділу методів та систем дистанційного зондування Фізико-механічного інституту НАН України вул.Наукова,5,м.Львів,79601,Україна.

Висоцька Вікторія Анатоліївна – к.т.н., доцент, доцент кафедри «Інформаційні системи та мережі» Національного університету «Львівська політехніка», вул. Степана Бандери 12, м. Львів, 79013, Україна.

Осипов Михайло Михайлович – магістр кафедри «Інформаційні системи та мережі» Національного університету «Львівська політехніка», вул. Степана Бандери 12, м. Львів, 79013, Україна.

Варецький Ярема Юрійович – к.т.н.,ст.н.с. відділу методів та систем дистанційного зондування Фізико -механічного інституту НАН України,вул. Наукова,5,м. Львів, 79601, Україна.

Капшій Олег Вірославович – к.т.н.,н.с. відділу методів та систем дистанційного зондування Фізико - механічного інституту НАН України, вул. Наукова,5,м. Львів, 79601, Україна.

B.P. Rusyn[1], L.V. Pohreliuk[1], V.A. Vysotska[2], M.M. Osypov[2],
J.Y. Varetsky[1], OV Kapshiy[1]

SYSTEM ARCHITECTURE OF DATA DEDUBLATION AND DISTRIBUTION IN CLOUD STORES DURING BACKUP

[1]GV Physical and Mathematical Institute Karpenko NAS of Ukraine, Lviv

[2]NU Lviv Polytechnic, Department of Information Systems and Networks, Lviv

Б.П. Русын[1], Л.В. Погрелюк[1], В.А. Высоцька[2], М.М. Осыпов[4],
Я.Ю. Варецькый[1], А.В. Капший[1]

АРХИТЕКТУРА СИСТЕМЫ ДЕДУБЛИКАЦИИ И РАСПРЕДЕЛЕНИЯ ДАННЫХ В ОБЛАЧНОЕ ХРАНИЛИЩЕ ПРИ РЕЗЕРВНОМ КОПИРОВАНИИ

[1]Физико-математический институт имени Г.В. Карпенко НАН Украины, Львов

[2]НУ «Львовская политехника», кафедра «Информационные системы и сети», Львов