МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Навчально-науковий інститут інноваційних освітніх технологій
Кафедра комп'ютерної інженерії

ХАРЧУН Василь Вікторови

Алгоритми надійного розподіленого зберігання даних на основі корегуючих кодів / Algorithms of the resistant distributed data storing based on correction codes

спеціальність: 123 - Комп’ютерна інженерія
магістерська програма - Комп’ютерна інженерія

Магістерська робота

Виконав студент групи КІм-21
В. В Харчун
Науковий керівник: д.т.н. професор,
В. В. Яцків

Магістерську роботу допущено до захисту:

ТЕРНОПІЛЬ -2018

РЕЗЮМЕ

Магістерська робота на тему «Алгоритми надійного розподіленого зберігання даних на основі корегуючих кодів» зі спеціальності 123 «Комп’ютерна інженерія» написана обсягом 83 сторінок і містить 29 ілюстрацій, 10 таблиць, 3 додатки та 51 джерел за переліком посилань.

Метою роботи є підвищення надійності систем зберігання даних на основі корегуючих кодів.

Методи досліджень. Для розв’язання поставлених задач у магістерській роботі використано: теорія інформації та кодування, методи завадостійкого кодування даних, теорію розподілених алгоритмів, імплементації цифрових систем на кристалі, мови опису апаратури.

Результати дослідження: алгоритм розподіленого зберігання даних, алгоритм відновлення даних, структура кодера/декодера.

Результати роботи можуть бути використані при проектуванні локальних та мережних систем зберігання даних а також в навчальному процесі.

Орієнтовні напрямки розвитку досліджень: підвищення швидкодії роботи алгоритмів відновлення даних з використанням корегуючих кодів.

КЛЮЧОВІ СЛОВА: АЛГОРИТМ ЗБЕРІГАННЯ ДАНИХ, КОРИГУЮЧІ КОДИ, КОДИ РІДА-СОЛОМОНА, СИСТЕМА ЗБЕРІГАННЯ ДАНИХ.

RESUME

Master's thesis: "Algorithms of the resistant distributed data storing based on correction codes" from the specialty 123 "Computer engineering" written 83 page volume and contains 29 illustrations, 10 tables, 3 applications and 51 sources for references.

The purpose of the work is to increase the reliability of data storage systems based on correction codes.

Research methods. To solve the tasks set in the master's work, we use: theory of information and coding, methods for preventing data coding, the theory of distributed algorithms, the implementation of digital systems on the crystal, the language of description of equipment.

Results of the study: distributed storage algorithm, data recovery algorithm, encoder / decoder structure.

The results of the work can be used in the design of local and network storage systems as well as in the learning process.

The approximate directions of research development: increasing the speed of data recovery algorithms using corrective codes.

KEY WORDS: DATA STORAGE ALGORITHM, CORRECT CODES, RID-SOLOMON CODES, DATA STORAGE SYSTEM.

ЗМІСТ

ВСТУП

Задача підвищення надійності зберігання інформації і одночасного збільшення продуктивності системи зберігання даних займає розуми розробників комп'ютерної периферії вже давно. Щодо підвищення надійності зберігання все зрозуміло: інформація - це товар, і нерідко дуже цінний. Для захисту від втрати даних придумано чимало способів, найбільш відомий і надійний з яких - це резервне копіювання інформації.

Питання підвищення продуктивності дискової підсистеми досить складне. Зростання обчислювальних потужностей сучасних процесорів привело до того, що спостерігається явний дисбаланс між можливостями жорстких дисків і потребами процесорів. Однак якщо не вистачає можливостей одного диска, то, можливо, частково вирішити дану проблему дозволить наявність декількох дисків. Звичайно, сама по собі наявність двох або більше жорстких дисків на комп'ютері або на сервері справи не змінює - потрібно змусити ці диски працювати спільно (паралельно) один з одним так, щоб це дозволило підвищити продуктивність дискової підсистеми на операціях запису / читання. Крім того, не можна, використовуючи декілька жорстких дисків, домогтися підвищення не тільки продуктивності, але і надійності зберігання даних, щоб вихід з ладу одного з дисків не призводив до втрати інформації? Саме такий підхід був запропонований ще в 1987 році американськими дослідниками Паттерсоном, Гібсоном і Катц з Каліфорнійського університету Берклі. У своїй статті «A Case for Redundant Arrays of Inexpensive Discs, RAID» («надлишковий масив недорогих дисків») вони описали, яким чином можна об'єднати кілька дешевих жорстких дисків в один логічний пристрій так, щоб в результаті підвищувалися ємність і швидкодія системи, а відмова окремих дисків не приводила до відмови всієї системи.

З моменту виходу статті пройшло вже більше 30 років, але технологія побудови RAID-масивів не втратила актуальності і сьогодні. Єдине, що

змінилося з тих пір, - це розшифровка абревіатури RAID. Справа в тому, що спочатку RAID-масиви будувалися зовсім не на дешевих дисках, тому слово Inexpensive (недорогі) поміняли на Independent (незалежні), що більше відповідало дійсності.

Більш того, саме зараз технологія RAID набула значного поширення. Так, якщо ще кілька років тому RAID-масиви використовувалися в дорогих серверах масштабу підприємства з застосуванням SCSI-дисків, то сьогодні вони стали своєрідним стандартом де-факто навіть для серверів початкового рівня. Крім того, поступово розширюється і ринок RAID-контролерів, тобто актуальність набуває завдання побудови RAID-масивів на робочих станціях з використанням дешевих дисків. Так, деякі виробники материнських плат (Abit, Gigabyte) вже почали інтегрувати RAID-контролери на самі плати.

Отже, RAID - це надлишковий масив незалежних дисків (Redundant Arrays of Independent Discs), на який покладається завдання забезпечення відмовостійкості і підвищення продуктивності. Відмовостійкість досягається за рахунок надлишковості. Тобто частина ємності дискового простору відводиться для службових цілей, стаючи недоступною для користувача.

Мета і завдання дослідження. Метою роботи є підвищення надійності систем зберігання даних на основі корегуючих кодів.

Досягнення визначеної мети передбачає вирішення таких завдань:

− провести аналіз підходів до надійного зберігання даних;

− розробити алгоритм зберігання даних на основі корегуючих кодів;

− розробити структуру системи зберігання даних на основі корегуючих кодів;

− реалізувати на програмованій логічній інтегральній схемі кодер обчислення перевірочних символів;

− розробити структуру пристрою відновлення даних;

− провести дослідження розроблених алгоритмів.

Об’єкт дослідження – процес надійного розподіленого зберігання даних на основі корегуючих кодів.

Предмет дослідження – алгоритми надійного розподіленого зберігання даних на основі корегуючих кодів.

Методи досліджень. Для розв'язання поставлених задач у магістерській роботі використано: теорія інформації та кодування, методи завадостійкого кодування даних, теорію розподілених алгоритмів, імплементації цифрових систем на кристалі, мови опису апаратури.

Наукова новизна одержаних результатів. Удосконалено алгоритм розподіленого зберігання даних на основі корегуючих кодів.

Практичне значення отриманих результатів. Розроблено структуру системи розподіленого зберігання даних на основі корегуючих кодів.

Публікації та апробація ДР. Матіїшин Ю.С., Харчун В.В., Гевко Я.І. Система надійного розподіленого зберігання даних на основі коригуючих кодів. Матеріали семінару комп'ютерні науки та інформаційні технології, CSIT'2018, Тернопіль, 2 червня 2018 р., С.48.

Впровадження результатів МР. Результати роботи прийняті до впровадження в Науково-дослідному інституту інтелектуальних комп'ютерних систем Тернопільського національного економічного університету.

Короткий зміст розділів. В першому розділі проведено аналіз підходів до надійного зберігання даних. Проведено аналіз та досліджено ефективність різних типів RAID масивів. Виділено переваги та недоліки різних способів реалізації RAID-масивів.

В другому розділі розглянуто теоретичні основи коригуючих кодів для системах надійного зберігання даних. Розроблена структура системи зберігання даних з використанням коригуючих кодів. Розроблено алгоритм кодування на основі кодів Ріда – Соломона.

В третьому розділі розглянуто способи кодування та декодування і схемна реалізація кодуючих пристроїв. Проведено дослідження корегуючих кодів в системах зберігання даних.

1 АНАЛІЗ СИСТЕМ НАДІЙНОГО ЗБЕРІГАННЯ ДАНИХ

1.1 Аналіз підходів до надійного зберігання даних

Підвищення продуктивності дискової підсистеми забезпечується одночасною роботою декількох дисків, і в цьому сенсі чим більше дисків в масиві (до певної межі), тим краще [26-28].

Спільну роботу дисків в масиві можна організувати з використанням або паралельного, або незалежного доступу.

При паралельному доступі дисковий простір розбивається на блоки (смужки) для запису даних. Аналогічно інформація, що підлягає запису на диск, розбивається на такі ж блоки. При запису окремі блоки записуються на різні дисків, причому запис декількох блоків на різні диски відбувається одночасно, що й призводить до збільшення продуктивності в операціях запису. Потрібна інформація також зчитується окремими блоками одночасно з декількох дисків, що також сприяє зростанню продуктивності пропорційної кількості дисків в масиві.

Слід зазначити, що модель з паралельним доступом реалізується тільки за умови, що розмір запиту на запис даних більше розміру самого блоку. В іншому випадку реалізувати паралельний запис декількох блоків просто неможливо. Уявімо ситуацію, коли розмір окремого блоку становить 8 Кбайт, а розмір запиту на запис даних - 64 Кбайт. В цьому випадку вихідна інформація нарізається на вісім блоків по 8 Кбайт кожен. Якщо є масив з чотирьох дисків, то одночасно можна записати чотири блоки, або 32 Кбайт, за один раз. Очевидно, що в розглянутому прикладі швидкість запису і швидкість зчитування виявиться в чотири рази вище, ніж при використанні одного диска. Однак така ситуація є ідеальною, оскільки далеко не завжди розмір запиту кратний розміру блоку і кількості дисків в масиві.

Якщо ж розмір записуваних даних менше розміру блоку, то реалізується принципово інша модель доступу - незалежний доступ. Більш того, ця модель

може бути реалізована і в тому випадку, коли розмір записуваних даних більше розміру одного блоку. При незалежному доступі всі дані окремого запиту записуються на окремий диск, тобто ситуація ідентична роботі з одним диском. Перевага моделі з паралельним доступом в тому, що при одночасному надходженні декількох запитів на запис (читання) всі вони будуть виконуватися незалежно, на окремих дисках. Подібна ситуація є типовою, наприклад, в серверах.

У відповідності з різними типами доступу існують і різні типи RAID-масивів, які прийнято характеризувати рівнями RAID. Крім типу доступу, рівні RAID розрізняються способом розміщення та формування надлишкової інформації. Надлишкова інформація може або розміщуватися на спеціально виділеному диску, або перемішуватися між усіма дисками. Способів формування цієї інформації дещо більше. Найпростіший з них - це повне дублювання (100-процентна надмірність), або віддзеркалення. Крім того, використовуються коди з корекцією помилок, а також обчислення парності.

Регулярне копіювання критично важливої інформації може убезпечити активи, але для живого бізнесу цього недостатньо, запорука його успіху - в безперервності. З такою двоєдиної ідеї (безпечної захисту даних на працюючих дискових накопичувачах при збереженні доступу до них) зросли RAID-масиви, а різноманітність призначених для користувача сценаріїв породило безліч рівнів RAID, список модифікацій яких продовжує поповнюватися. Перед тим як перейти до опису «нових надходжень», згадаємо про причини подібного різноманітності.

Незалежно від призначення RAID користувач знаходиться в тривимірному просторі прийняття рішення, де по осях відкладені вимоги до продуктивності, захищеності (доступності) даних і ціна рішення. Навіть при адекватній оцінці доцільність цілям можливі варіанти, що вже говорити про логіку вибору відповідної реалізації RAID при помилках щодо вагомості перерахованих факторів. І наведена в статті таблиця в найзагальніших рисах описує його мотиви і критерії.

Коли мова заходить практично про будь-яку частину комп'ютера, можна сказати, що було б добре, щоб вона працювала швидше і надійніше. Це особливо відноситься до підсистем, від яких загальна продуктивність комп'ютера залежить найбільшою мірою. І звичайно, це відноситься до його дискової підсистеми.

Дискових операцій в більшості комп'ютерів буває дуже багато. Адже комп'ютери призначені для обробки даних, які потрібно десь зберігати, і потрібно кудись записувати результат їх обробки.

З 1956 року, коли з'явилася перша модель жорсткого магнітного диска, деякі характеристики дисків покращилися фантастично. Наприклад, щільність запису зросла в 650 000 000 разів.

А ось час доступу до даних скоротилося всього в сотні разів. Навіть якщо в тисячу - все одно прогрес по швидкості доступу виявився набагато скромнішим.

Уже в 70-ті роки виникла ідея розділяти дані (наприклад, файли) на фрагменти і записувати їх на різні диски одночасно. Очевидно, що загальний час запису всього файлу скоротиться. Теоретично виграш може бути в стільки разів, скільки дисків задіяно для паралельного запису. На рисунку 1.1 для наочності чергування показано на прикладі файлу [26].

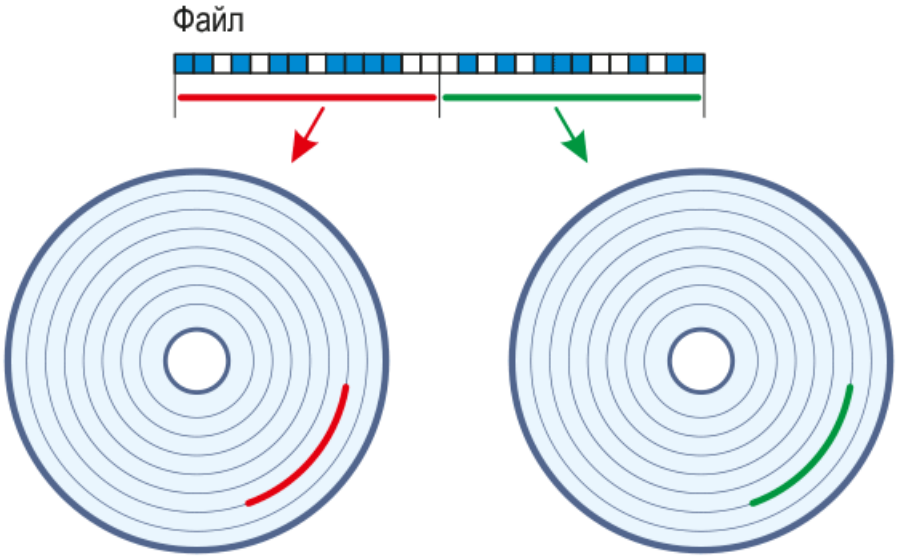


Рисунок 1.1 – Приклад паралельного запису файлу

Коли записані дані будуть потрібні, відповідні їм фрагменти потрібно прочитати з різних дисків (бажано також одночасно), а потім зібрати в загальний блок.

Цей прийом запису на диск називається чергуванням (striping).

Насправді чергуються дискові одиниці зберігання: блоки, сектора, сторінки. Файлова система знаходиться на більш високому архітектурному рівні.

У описаного методі прискорення дискових операцій є дуже суттєвий недолік. Якщо хоча б один з фрагментів буде втрачено або зіпсований, весь блок даних так само виявиться втраченим.

Надійніше. Час напрацювання на відмову сучасних дисків досягає 2 500 000 годин, ... 285 років безперервної роботи. Але це - статистика, а конкретний диск може зламатися і через годину після включення. Так, ймовірність такої поломки дуже-дуже мала, але вона - не нульова. Тому навіть при використанні супернадійних дисків не гріх застосувати додаткові заходи для забезпечення збереженості даних. Стандартний спосіб запобігання втрати даних - їх резервне копіювання, тобто дублювання.

Наприклад, копію даних, що записуються на один диск, можна одночасно записувати на інший (рисунок 1.2).

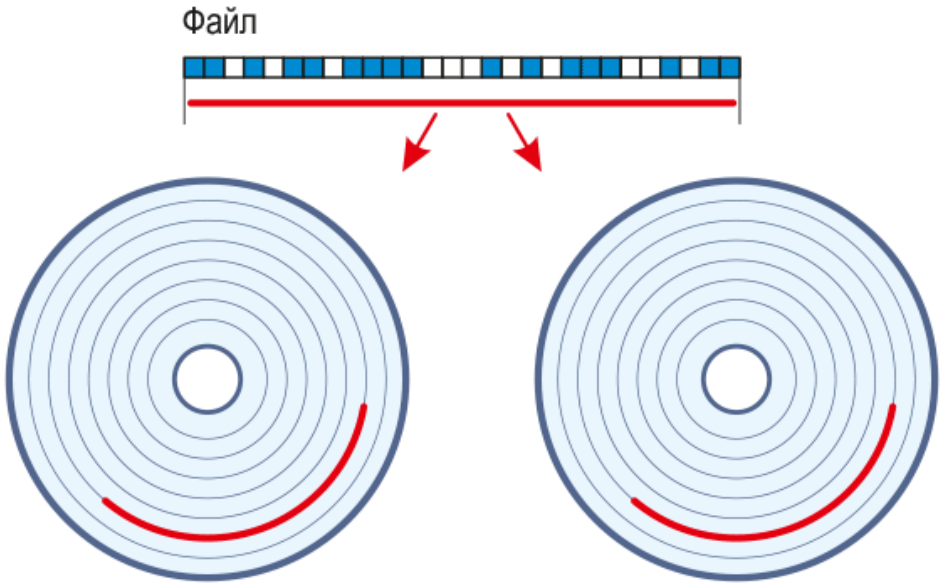


Рисунок 1.2 – Дублювання даних

У разі збою першого диску можна буде прочитати дані з другого диску. Втім, спочатку потрібно вчасно зрозуміти, що збій стався. Для цього є спеціальні методи.

Описаний спосіб дискових операцій називається дзеркалюванням (mirroring).

На рисунку 1.2 для наочності віддзеркалення показано на прикладі файлу. Насправді віддзеркалюються дискові одиниці зберігання: блоки, сектора, сторінки.

І у цього методу теж є один дуже суттєвий недолік: для його реалізації потрібно дискового простору в два рази більше, ніж для зберігання даних без віддзеркалення.

І швидше, і надійніше. А як можна забезпечити одночасне підвищення і швидкості дискових операцій, і їх надійності? – Можна, але доведеться використовувати більше двох дисків.

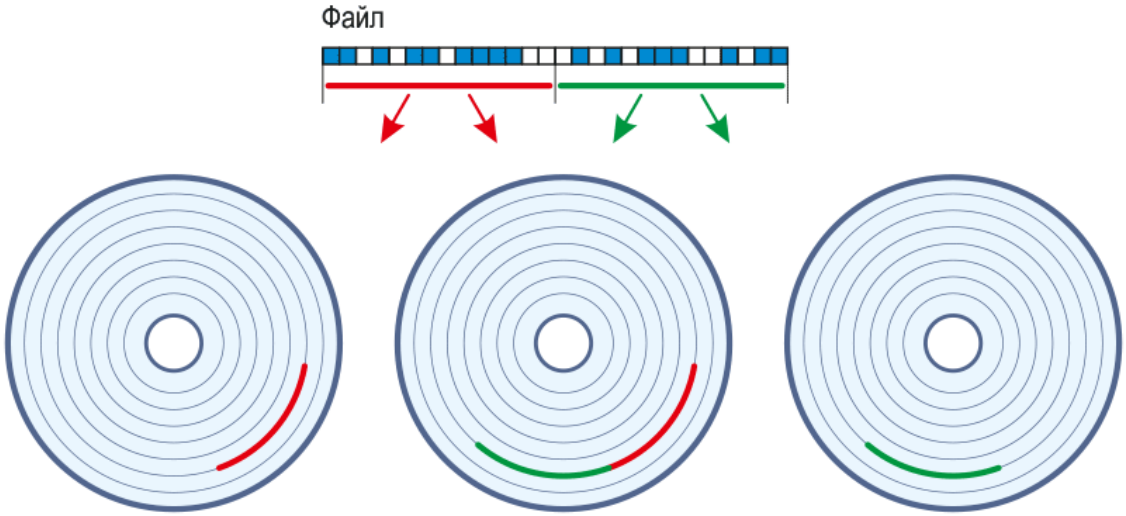


Рисунок 1.3 – Приклад запису на три диски

Застосувати в дисковій підсистемі можна і більше трьох дисків. Тоді ефективність зберігання даних зросте, а прояв зазначених недоліків скоротиться.

Типи RAID-масивів. Описані методи організації дискової підсистеми розвиваються вже давно. Вони досить докладно описані і стандартизовані.

Групи спільно використовуваних дисків називають RAID-масивами. RAID - Redundant Array of Independent Disks, надлишковий масив незалежних дисків.

RAID-масиви класифікують за умовними рівнями. Чергування в базовому вигляді реалізовано в RAID-масивах нульового рівня (RAID 0).

Віддзеркалення в базовому вигляді реалізовано в RAID-масивах першого рівня (RAID 1). Як вже було сказано, в групі може бути присутнім більше двох дисків. Це дасть можливість скористатися перевагами як чергування, так і віддзеркалення і одночасно нівелювати їх недоліки. Поєднання цих методів реалізовано в RAID 2, RAID 3, RAID 4, RAID 5, RAID 6 і далі.

1.2 Типи RAID масивів

Масив RAID 0. Дисковий масив підвищеної продуктивності без відмовостійкості (Striped Disk Array without Fault Tolerance) (рисунок 1.4).

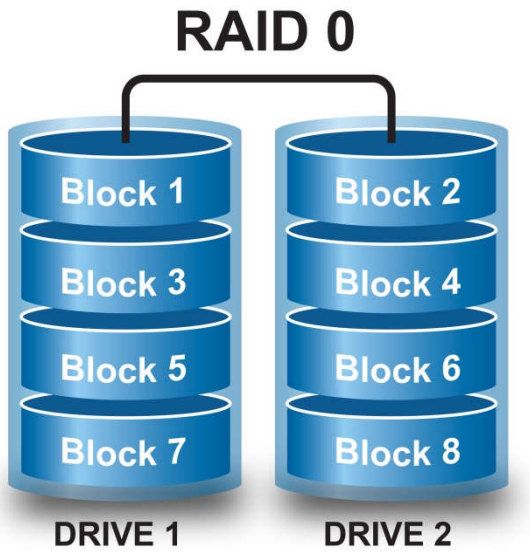


Рисунок 1.4 – Структура масиву RAID 0.

Масив RAID 0 найбільш продуктивний і найменш захищений з усіх RAID масивів. Дані розбиваються на блоки пропорційно кількості дисків, що призводить до більш високої пропускної здатності. Висока продуктивність даної структури забезпечується паралельним записом і відсутністю надлишкового копіювання. Відмова будь-якого диска в масиві призводить до втрати всіх даних. Цей рівень називається striping.

Переваги [27]:

– найвища продуктивність для додатків, які потребують інтенсивної обробки запитів вводу / виводу і даних великого обсягу;

– простота реалізації;

– низька вартість на одиницю об'єму.

Недоліки:

– не поломок рішення;

– відмова одного диску тягне за собою втрату всіх даних масиву.

Масив RAID 1. Дисковий масив з дублюванням або віддзеркалення Duplexing & Mirroring. RAID 1 - mirroring - дзеркальне відображення двох дисків. Надлишковість структури даного масиву забезпечує його високу відмовостійкість (рисунок 1.5).

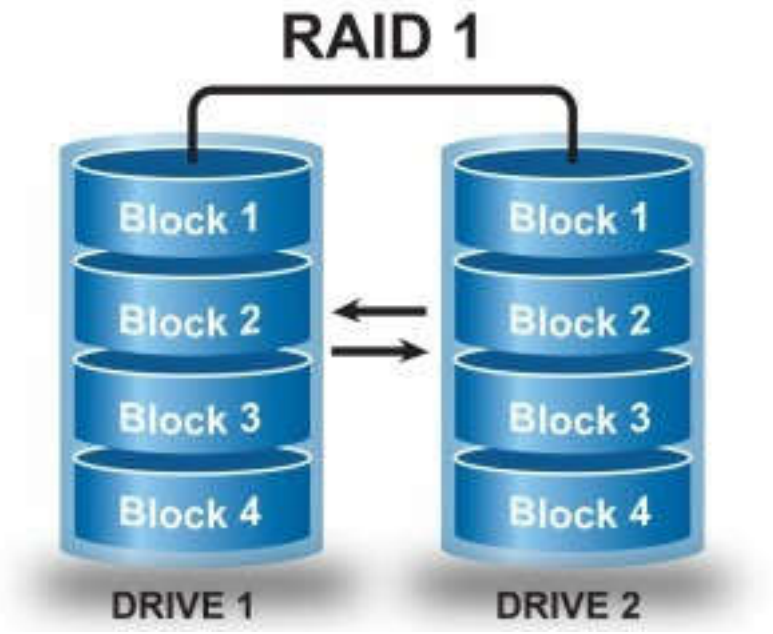


Рисунок 1.5 – Структура масиву RAID 1

Масив відрізняється високою собівартістю і низькою продуктивністю.

Переваги:

– простота реалізації;

– простота відновлення масиву в разі відмови (копіювання);

– досить висока швидкодія для додатків з великою інтенсивністю запитів.

Недоліки:

– висока вартість на одиницю об'єму - 100% надлишковість;

– невисока швидкість передачі даних.

Масив RAID 2. Відмовостійкий дисковий масив з використанням коду Хеммінга (Hamming Code, ECC). RAID 2 - використовує коди виправлення помилок Хеммінга (Hamming Code, ECC) (рисунок 1.6).

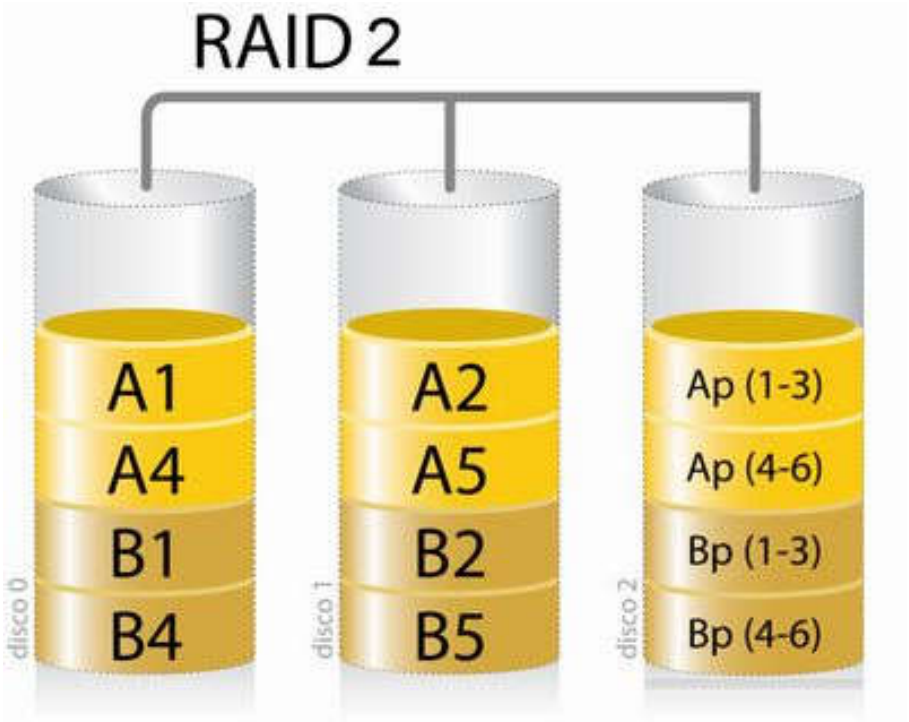


Рисунок 1.6 – Структура масиву RAID 2

Коди дозволяють виправляти одиночні і виявляти подвійні помилки.

Переваги:

– швидка корекція помилок ("на льоту");

– дуже висока швидкість передачі даних великих обсягів;

– при збільшенні кількості дисків, накладні витрати зменшуються;

– досить проста реалізація.

Недоліки:

– висока вартість при малій кількості дисків;

– низька швидкість обробки запитів (не підходить для систем орієнтованих на обробку транзакцій).

Опис RAID 3. Відмовостійкий масив з паралельною передачею даних і перевіркою на парність Parallel Transfer Disks with Parity. RAID 3 - дані зберігаються за принципом striping на рівні байтів з контрольною сумою (КС) на одному з дисків (рисунок 1.7). Масив не має проблем з надлишковістю, як в RAID 2-го рівня. Диски з контрольною сумою використовувані в RAID 2, необхідні для визначення помилкового заряду. Однак більшість сучасних контролерів здатні визначити, коли диск відмовив за допомогою спец сигналів або додаткового кодування інформації, записаної на диск і використовуваної для виправлення випадкових збоїв.

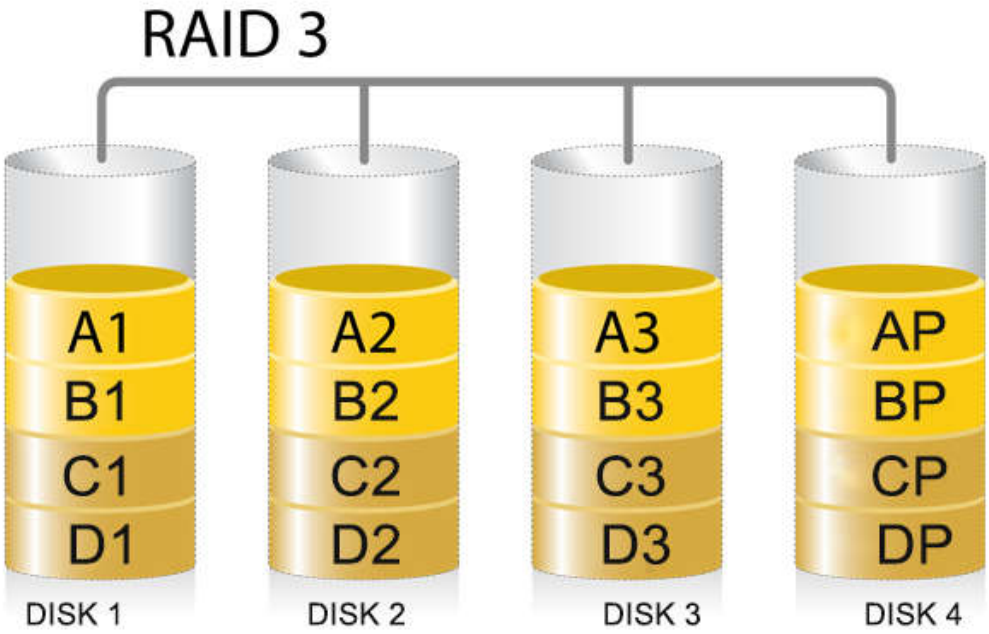


Рисунок 1.7 – Структура масиву RAID 3

Переваги:

– дуже висока швидкість передачі даних;

– відмова диска мало впливає на швидкість роботи масиву;

– малі накладні витрати для реалізації надлишковості.

Недоліки [28]:

– складна реалізація;

– низька продуктивність при великій інтенсивності запитів даних невеликого об'єму.

RAID 4. RAID 4 заснований на принципі RAID 3, але відрізняється розбивкою даних не на байти, а на блоки. Таким чином, вирішується проблема низької швидкості передачі даних малих обсягів. Запис же проводиться повільно через те, що парність блоку генерується під час запису і записується на єдиний диск (рисунок 1.8).

Використання. RAID 4 раніше використовувався в дискових масивах NetApp серії FAS, де недолік надійності компенсувався принципом записи файлової системи WAFL (Write Anywhere File Layout). Рекомендується тільки для тимчасових і неважливих даних. На даний момент RAID 4 в МСД NetApp витісняється RAID DP.

Відповідно до рекомендацій компанії NetApp, оптимальний розмір RAID групи для NL-SAS становить 7 дисків; для SAS або SSD - від 8 до 14 дисків.

Переваги.

– значне підвищення швидкості читання в порівнянні з дисками, які не об'єднані в RAID;

– висока ефективність дискового простору в порівнянні з RAID 1, 6, DP, 10;

– базовий рівень надійності, допустимо вихід з ладу одного диска.

Недоліки:

– високе навантаження на диск парності і, як наслідок, зниження терміну експлуатації (за винятком використання в системах NetApp серії FAS);

– в разі виходу з ладу диска парності, процес відновлення (rebuild) RAID-групи різко підвищує навантаження на диски, що може спровокувати вихід другого диска всередині групи і привести до втрати даних.

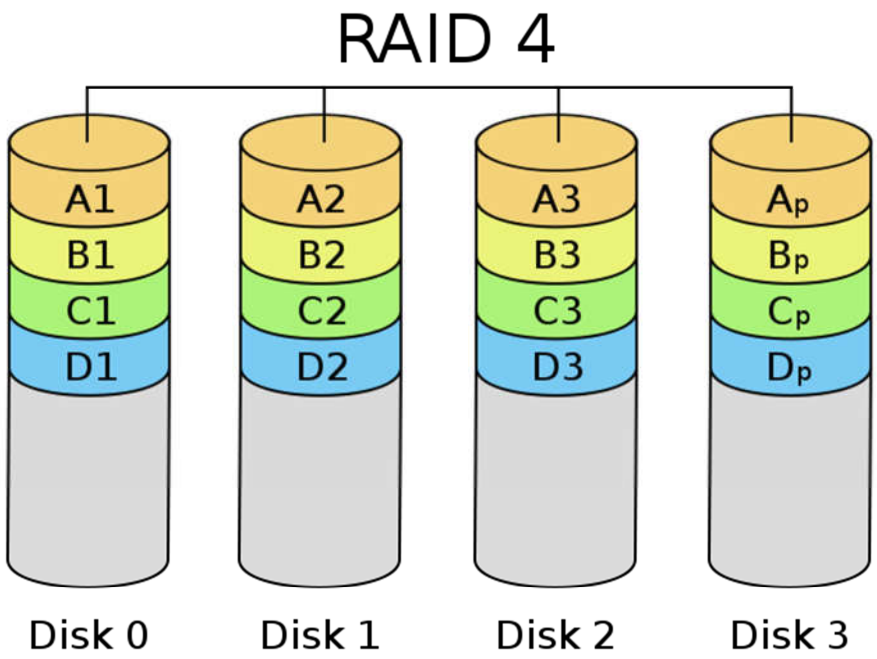


Рисунок 1.8 – Структура масиву RAID 4

RAID 5. RAID 5 не використовує для зберігання інформації про парність окремий контрольний диск. Блоки даних і контрольні суми циклічно записуються на всі диски масиву, при цьому немає асиметричності конфігурації дисків (рисунок 1.9). RAID 5 рівня отримав поширення в системах середнього рівня через свою дешевизну. Розмір дискового простору масиву RAID 5 розраховується за формулою (n-1) * hdsize, де n - число дисків в масиві, а hddize - розмір одного диска. Наприклад для масиву з 4-х HDD по 200 гігабайт загальний обсяг буде (4-1) * 200 = 600 гігабайт. Недоліки масиву RAID 5 проявляються при виході з ладу одного з дисків, коли весь том переходить в критичний режим, всі операції запису і читання супроводжуються додатковими маніпуляціями, різко падає продуктивність. При цьому рівень надійності знижується до надійності RAID-0 з відповідною кількістю дисків. Якщо до повного відновлення масиву відбудеться вихід з ладу хоча б ще одного диска, то масив руйнується, і дані на ньому відновленню звичайними методами не підлягають. У масиві RAID 5 можна використовувати диск з технологією HotSpare, яка полягає в тому, що основний час цей диск простоює, але при

виході з ладу одного з дисків масиву він включається в роботу і негайно починає відновлення.

Використання. RAID 5 широко застосовується в реальних бізнес-задачах. Типовим сценарієм є використання в серверах в області зберігання даних для призначених для користувача додатків і виконання транзакцій. Використовувати в СГД рекомендується в першу чергу для маловажной інформації з невеликим навантаженням на диски, наприклад, в системах відеоспостереження.

Переваги:

– значне підвищення швидкості читання в порівнянні з дисками, які не об'єднаними в RAID;

– висока ефективність використання дискового простору в порівнянні з RAID 1, 6, DP, 10;

базовий рівень надійності, припустимо вихід з ладу одного диска.

недоліки:

– в разі виходу з ладу одного диска, процес відновлення (rebuild) RAID-групи різко підвищує навантаження на диски, що може спровокувати вихід другого диска всередині групи і привести до втрати даних;

– незначне зниження швидкості запису в IOPS в порівнянні з RAID 0, DP, 10.

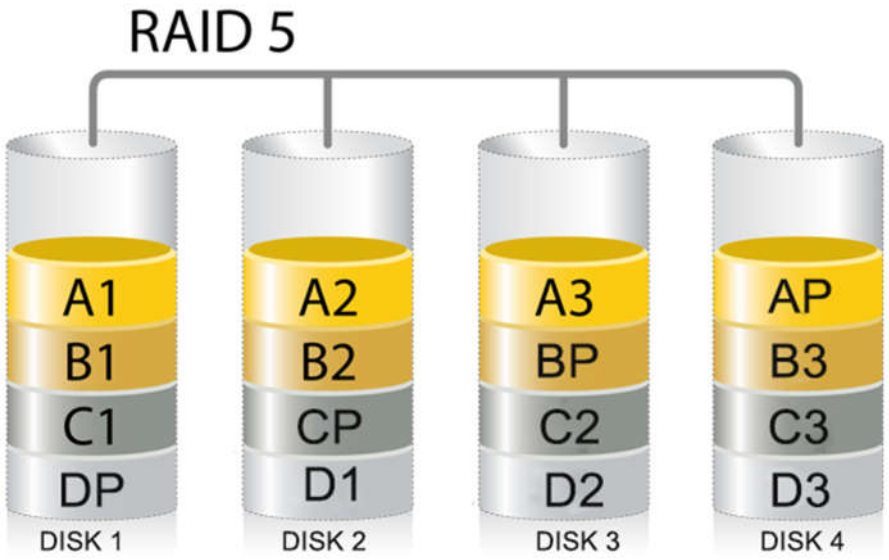


Рисунок 1.9 – Структура масиву RAID 5

RAID 6. RAID-масив рівня 6 по технології схожий на RAID 5 рівня, однак відрізняється підвищеним рівнем надійності, тому що під контрольні суми виділяється два жорсткі диски. Для правильної роботи RAID 6 необхідний потужний RAID-контролер. Захищає від так званої "кратної відмови", але для роботи вимагає, як мінімум 4 жорстких диска (рисунок 1.10).

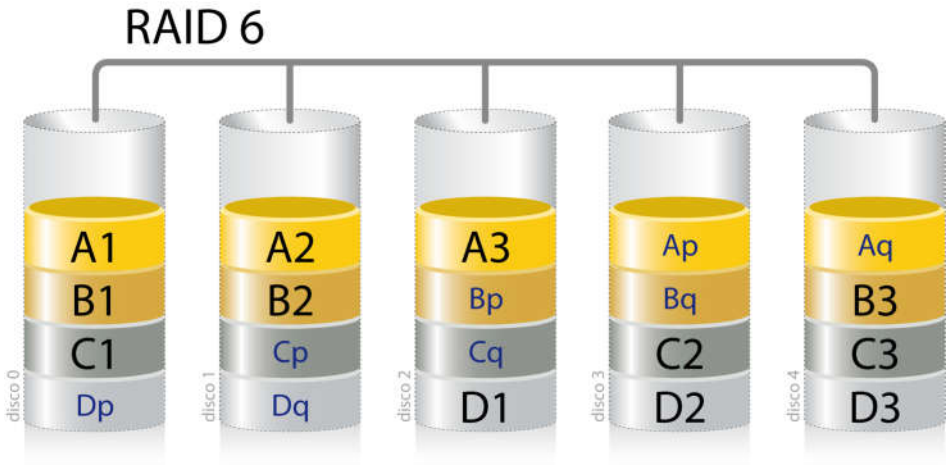


Рисунок 1.10 – Структура масиву RAID 6

RAID 6 використовує масив дисків з поблочним чергуванням з двома контрольними сумами. Дані розподілені по дискам масиву по черзі, в якості інформації для відновлення використовується схема подвійної парності. RAID 6 може витримати відмову двох дисків одночасно, однак низька продуктивність за операціями введення-виведення (IOPS) обмежує область застосування.

Використання. Сценарії застосування RAID 6 аналогічні RAID 5 з уклоном в більш надійне зберігання інформації. RAID 6 широко застосовується в системах зберігання даних, де не важлива висока транзакційна продуктивність - архівне зберігання, відеоспостереження стратегічних об'єктів, використання в системах безпеки, а також для надійного зберігання критично важливих даних.

Переваги:

– значне підвищення швидкості читання в порівнянні з дисками, не об'єднаними в RAID;

– висока ступінь надійності в порівнянні з RAID 5, допустимий вихід з ладу двох дисків.

Недоліки:

– низька швидкість запису в IOPS;

– ефективність використання дискового простору нижче, ніж у RAID 5.

RAID 7. RAID 7 є розробка компанії SCC - Storage Computer Corporation. Структура масиву наступна: на n - 1 HDD зберігається інформація, один диск використовується для складування блоків парності (рисунок 1.11). Однак, є кілька важливих деталей, покликаних ліквідувати головний недолік масивів даного типу: кеш даних і потужний контролер, який відповідає за обробку запитів. Тим самим знижується кількість звернень до HDD для обчислення контрольної суми, а як наслідок збільшується швидкість обміну даними в 5-6 разів. З мінусів подібної системи звичайно ж висока вартість обладнання для побудови таких масивів.

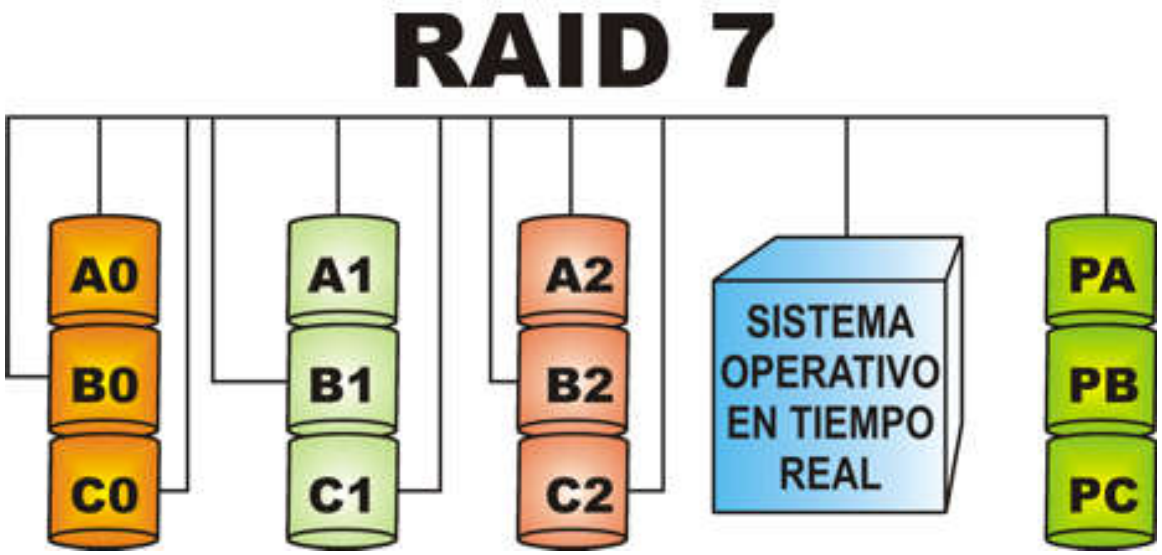


Рисунок 1.11 – Структура масиву RAID 7

RAID DP. RAID DP - масив дисків з подвійною парністю в лінійці продукції FAS компанії NetApp. Під контрольні суми виділяються два окремих

диска. Завдяки використанню файлової системи WAFL транзакційна продуктивність вище, ніж в RAID 5 і RAID 6.

Використання. Є основним типом RAID в обладнанні NetApp лінійки FAS. Рекомендується для SSD дисків, SATA / NL-SAS дисків об'ємом менш 6TB (для дисків більшого обсягу рекомендується RAID-TEC).

Відповідно до рекомендацій компанії NetApp, оптимальний розмір RAID групи - від 14 до 20 дисків.

Переваги:

– висока швидкість читання в порівнянні з дисками, які не об'єднаними в RAID;

– висока швидкість запису в IOPS в порівнянні з RAID 5 і RAID 6, наближається до RAID 10;

– високий ступінь надійності в порівнянні з RAID 5, припустимо вихід з ладу двох дисків.

Недоліки:

– даний RAID використовується тільки в дискових масивах NetApp серії FAS;

– ефективність використання дискового простору нижче ніж у RAID 5.

RAID TEC. RAID TEC - масив дисків з потрійною парністю в лінійці продукції FAS компанії NetApp. Даний тип RAID розроблений для повільних об'ємних дисків зі стійкою продуктивністю під час відновлення.

Використання. Рекомендується для SATA / NL-SAS дисків об'ємом 6TB і більш.

Відповідно до рекомендацій компанії NetApp, оптимальний розмір RAID групи - від 20 до 29 дисків.

Переваги:

– висока швидкість читання в порівнянні з дисками, які не об'єднаними в RAID;

– найвища ступінь надійності в поєднанні з хорошою ефективністю зберігання (близько 89% при RAID групі в 29 дисків).

Недоліки:

– даний RAID використовується тільки в дискових масивах NetApp серії FAS.

RAID 10. RAID 10 - масив дисків з дзеркалюванням і чергуванням. Являє собою масив RAID 0 з декількох масивів RAID 1. У реалізації RAID 10 на практиці кожен підмасив RAID 1 складається з двох дисків, тому допускається вихід з ладу не більше одного диска в кожному підмасиві. RAID 10 має найвищу продуктивність і надійність, при цьому ефективність використання дискового простору складає 50%.

Використання. Основним сценарієм застосування є використання для роботи з базами даних (Oracle, SAP HANA, SQL) і іншими високо транзакційними навантаженнями.

Переваги:

– найвища швидкість читання і запису серед комерційно використовуваних типів RAID;

– підвищена надійність в порівнянні з RAID 5.

Недоліки:

– ефективність використання дискового простору 50%.

RAID 50. RAID 50 - масив дисків, що складається з чергування масивів RAID 5. Реалізується шляхом побудови потоку (RAID 0) з RAID 5. Допускається відмову не більше одного диска в кожному підмасиві. Продуктивність RAID 50 вище, ніж при використанні RAID 5, і прагне до RAID10, але надійність недостатня для застосування в реальних бізнес-задачах.

Використання. Підтримується обмеженою кількістю виробників, так як не рекомендується використовувати із-за низької надійності. Можливо модель використання передбачає зберігання тимчасових або неважких даних.

Переваги [26, 27]:

– значне підвищення швидкості читання в порівнянні з дисками, не об'єднаними в RAID;

– висока ефективність використання дискового простору порівняно з RAID 1, 6, DP, 10.

Недоліки:

– незначне зниження швидкості запису в IOPS в порівнянні з RAID 0, DP, 10;

– недостатня надійність для комерційного використання.

RAID 60 - масив дисків, що складається з чергування масивів RAID 6. Реалізується побудовою Страйп (RAID 0) з RAID 6. Допускається відмова до двох дисків в кожному підмасиві. Володіє базовою надійністю і невисокою ефективністю використовуваного простору. Для побудови мінімальної RAID-групи потрібно 8 дисків (таблиця 1.1).

Використання. Підтримується обмеженою кількістю виробників, тому що не володіє явними перевагами в порівнянні з використанням інших типів RAID. Область практичного застосування обмежена.

Таблиця 1.1 – Порівняння ефективності RAID масивів

Тип RAID масиву	Ефективність
RAID 0	I=S * N
RAID 1	I=S
RAID 2	I=S * (N - 1)
RAID 3	I=S * (N - 1)
RAID 4	I=S * (N - 1)
RAID 5	I=S * (N - 1)
RAID DP	I=S * (N - 2)
RAID TEC	I=S * (N - 3)
RAID 10	I=S * N / 2
RAID 50	I=S * (N - 2)
RAID 60	I=S * (N - 4)

де N – кількість дисків в масиві, S – об'єм найменшого диска.

Переваги:

– значне підвищення швидкості читання в порівнянні з дисками, які не об'єднаними в RAID;

– базова надійність зберігання.

Недоліки:

– невисока ефективність використання дискового простору в порівнянні з RAID 5, 6, DP;

– швидкість запису в IOPS нижче, ніж у RAID 0, DP, 10.

Реалізації RAID. У літературі пропонується більше десятка рівнів RAID, на практиці використовується половина з них, а користуються найчастіше трьома: RAID 1 (з дзеркалюванням даних), почасти RAID 10 (з розподілом віддзеркалювати даних) і, зрозуміло, RAID 5 (з розподілом даних і контрольних сум, КС). Популярність RAID 5 можна вважати торжеством компромісу над обережністю - жоден з інших типів не дає такого високого ступеня використання доступного дискового простору.

Пряма економія на дисках при купівлі обладнання завжди переважує страхи перед можливою втратою даних. Але чи то розвиток суміжних технологій і продуктів (таких як HDD високої ємності і конструктивні елементи дискових масивів), то чи природне подорожчання накопичуваної інформації зіграло свою роль, тільки в останні роки отримали популярність нові рівні RAID. Загальна ідея їх появи на світ - збереження якомога вищої доступності даних і продуктивності деградованого масиву, що знаходиться в процесі відновлення після відмови одного з дисків (так званий rebuild).

На відміну від рівнів RAID «першої хвилі», що описують алгоритми обробки одиночних помилок, тут основна увага приділяється зниженню ризиків втрати даних від повторних збоїв і скорочення часу відновлення. Йдеться про RAID 1E, RAID 5E, RAID 5EE, RAID 6. Буква E (Enhanced) тут трактується, як розширення відповідних рівнів RAID для підвищення «боєздатності» дискового масиву і продовження терміну його служби.

1.3 Способи реалізація RAID-масивів

Для того, щоб почати використовувати групу дисків спільно, їх потрібно якось об'єднати і управляти ними. Це можна забезпечити двома способами: апаратно або програмно. У першому випадку використовується спеціальний «апаратний» контролер; у другому - масивом управляє спеціальна програма.

Багато сучасних операційних системи: Windows, macOS, Linux, FreeBSD та інші підтримують програмну реалізацію RAID-масиву [28].

Апаратний контролер бере на себе частину навантаження, пов'язаної з дисковими операціями, але вимагає додаткової витрати на своє придбання.

Апаратний. Найчастіше застосовується в додатках з малим розміром блоку - як в транзакційних базах даних, так і Web-серверах. Центральний процесор розвантажений від виконання ресурсномістких операцій. Кещування зі зворотним записом (при наявності акумуляторної захисту вмісту кеш-пам'яті) значно підвищує продуктивність, без ризику втрати даних.

Переваги. Захист даних і продуктивність; високу швидкодію і стійкість до помилок в порівнянні з програмним RAID.

Зовнішній. Підключається до сервера через стандартний контролер. Функції RAID виконує мікропроцесор зовнішнього, незалежного від хоста RAID-контролера.

Переваги. ОС-незалежність; дозволяє будувати високоємні системи зберігання, пов'язані з серверами.

При програмній реалізації RAID-масиву купувати додаткові апаратні засоби не потрібно, але додаткове навантаження лягає на операційну систему. У певному сенсі, обробка цього навантаження - теж витрата, тому що вам потрібно виділити на забезпечення роботи програмного контролера зовсім безкоштовні обчислювальні ресурси.

Програмний RAID масив. Зазвичай використовується в додатках з великим розміром блоку даних - таких як закачування сховищ даних або

перегляд відео в реальному часі. А також в серверах з достатніми ресурсами центрального процесора для обслуговування інтенсивних операцій введення / виводу в деяких рівнях RAID. Включений в ОС (Windows, Netware, Linux). Всі функції RAID реалізовані центральним процесором, що істотно обмежує його здатність до виконання інших обчислень. Переваги. Низька ціна; достатньо стандартного (наприклад, вбудованого) контролера.

2 КОРИГУЮЧІ КОДИ В СИСТЕМАХ НАДІЙНОГО ЗБЕРІГАННЯ ДАНИХ

2.1 Коригуючі коди Ріда – Соломона

Розглянемо використання кодів Ріда-Соломона в контексті систем зберігання даних. Почнемо з простих прикладів, які дозволять краще усвідомити сутність вирішуваних задач [29, 31-34].

Згаданий вище «алгебраїчний» підхід полягає в наступному. Складається матриця спеціального виду розміру 5×3. Перші три рядки цієї матриці утворюють одиничну матрицю, а інші два - це деякі числа, про вибір яких ми поговоримо пізніше. В англомовній літературі цю матрицю зазвичай називають generator matrix (породжуюча матриця). Помножимо сформовану матрицю на вектор, складений з вихідних чисел A, B і C (рисунок 2.2).

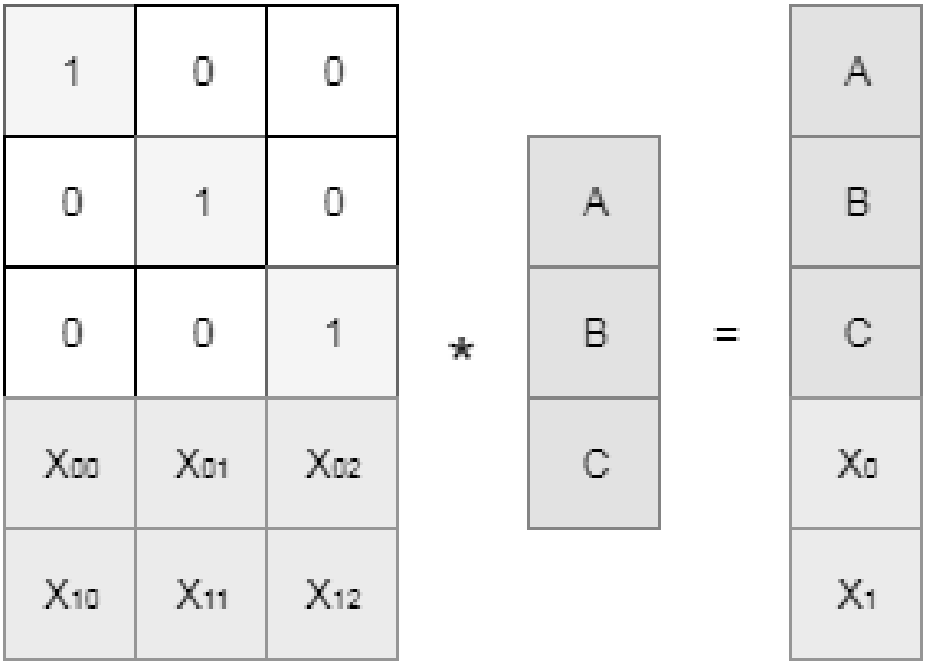


Рисунок 2.2 – Формування перевірочних символів

В результаті множення матриці на вектор з даними отримуємо два «надлишкових» числа, позначених на рисунку 2.2 як X_0 і X_1. Давайте

подивимося, як за допомогою цих «надлишкових» даних можна відновити, наприклад, втрачені A і B.

Викреслимо з породжуючої матриці рядки, які відповідають «зниклим» даним. У нашому випадку A відповідає першому рядку, а B - другий. Отриману матрицю помножимо на вектор з даними, і в результаті отримаємо наступне рівняння (рисунок 2.3).

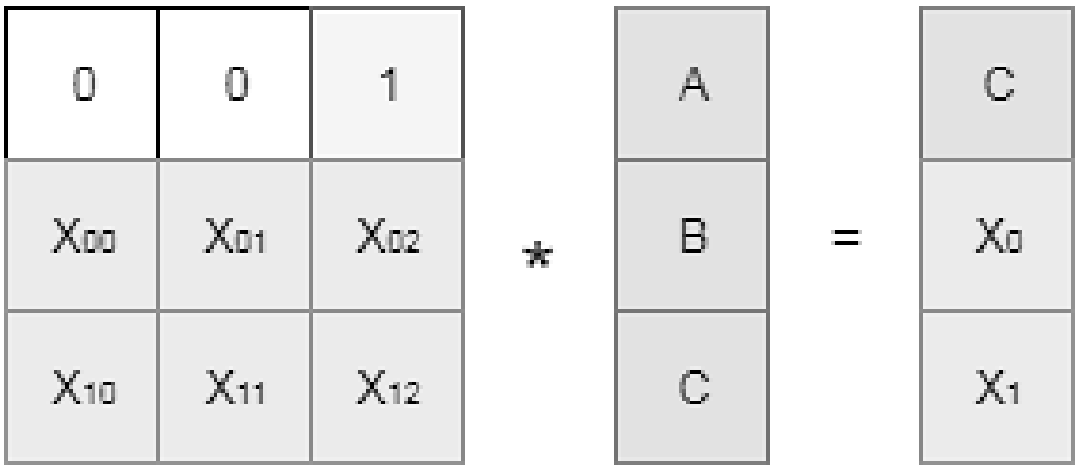


Рисунок 2.3 – Загальна схема відновлення даних

Проблема лише в тому, що A і B ми втратили, і вони нам наразі невідомі. Рішення цієї проблеми наступне.

Викреслимо відповідні рядки з породжуючої матриці і знайдемо зворотну до неї. На рисунку 2.4 ця зворотна матриця позначена як $\{Y_{ij}\}$. Тепер домножимо ліву і праву частини вихідного рівняння на зворотну матрицю (рисунок 2.4).

Скорочуючи матриці в лівій частині рівняння (добутком оберненої і прямої матриць є одинична матриця), і з огляду на той факт, що в правій частині рівняння немає невідомих параметрів, отримуємо вирази для шуканих A і B (рисунок 2.5).

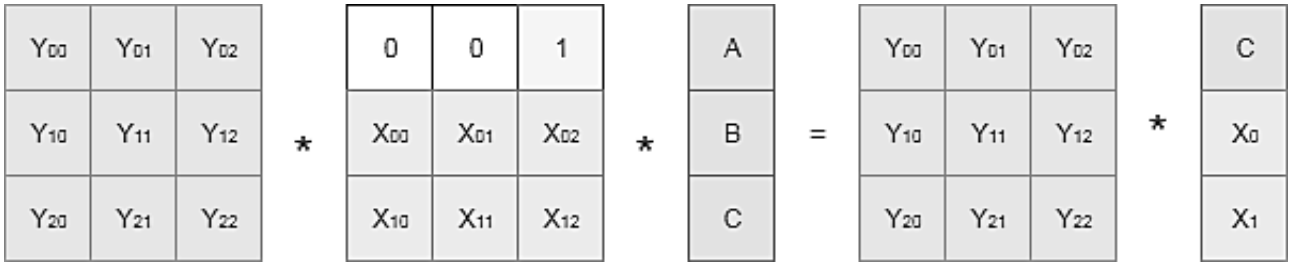


Рисунок 2.4 – Відновлення даних з використанням оберненої матриці

Власне кажучи, те, що ми зараз зробили - основа всіх типів кодів Ріда-Соломона, що застосовуються в системах зберігання даних. Процес кодування полягає в знаходженні «надлишкових» даних X_0, X_1, а процес декодування - в знаходженні оберненої матриці та множення її на вектор «збережених» даних.

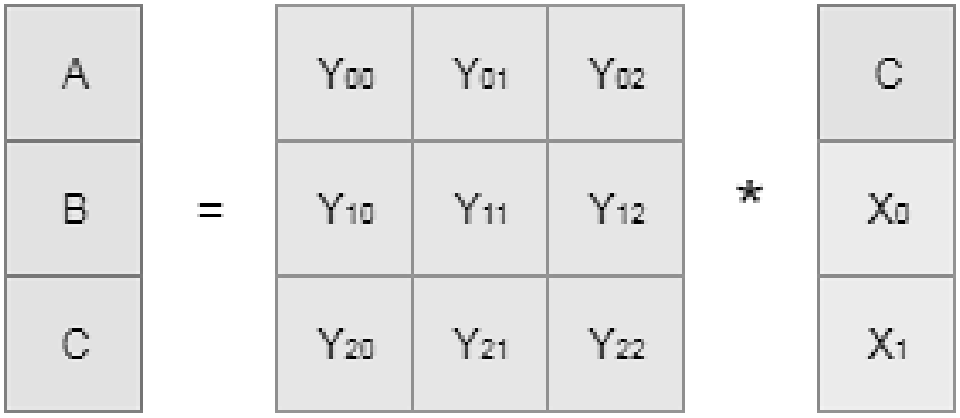


Рисунок 2.5 – Процес декодування

Неважко помітити, що розглянута схема може бути узагальнена на будь-яку кількість «вихідних» і «надлишкових» даних. Іншими словами, по вихідним N числах можна побудувати K надлишкових, причому завжди можливо відновити втрату будь-яких K з $N + K$ чисел. В цьому випадку породжуюча матриця буде мати розмір $(N + K) \times N$, а верхня частина матриці розміром $N \times N$ буде одиничною [4, 5].

Повернемося до питання про побудову породжуючої матриці. Чи можемо ми вибрати числа довільним чином? Відповідь - ні. Вибирати їх потрібно таким чином, щоб незалежно від рядків, які ми викреслюємо, матриця залишалася оберненою. В математиці відомі типи матриць, що володіють потрібним нам

властивістю. Наприклад, матриця Коші. У цьому випадку сам метод кодування часто називають методом Коші-Ріда-Соломона (Cauchy-Reed-Solomon). Іноді, для цих же цілей використовують матрицю Вандермонда, і відповідно, метод носить назву Вандермонда-Ріда-Соломона (Vandermonde-Reed-Solomon).

Переходимо до наступної задачі. Для представлення чисел в ЕОМ використовується фіксоване число байтів. Відповідно, в наших алгоритмах ми не можемо вільно оперувати довільними раціональними, і тим більше, речовими числами. Ми просто не зможемо реалізувати такий алгоритм на ЕОМ. У нашому випадку породжуюча матриця складається з цілих чисел, але при зверненні цієї матриці можуть виникнути раціональні числа, представляти які в пам'яті ПК проблематично. Для вирішення даної задачі використовують поля Галуа.

Поля Галуа. Ми всі звикли оперувати (додавати, віднімати, множити, ділити та інше) з числами - натуральними, раціональними, дійсними. Однак, замість звичних чисел, ми можемо почати розглядати довільну множину об'єктів (кінцеві і / або нескінченні) і вводити на цих множинах операції, аналогічні додаванню, відніманню і т.д. Власне кажучи, математичні структури типу груп, кілець, полів - це множини, на яких введені певні операції, причому, результати цих операцій знову належать вихідній множині. Наприклад, на множині натуральних чисел, ми можемо ввести стандартні операції додавання, віднімання і множення. Результатом знову буде натуральне число. А ось з розподілом все гірше, при розподілі двох натуральних чисел результат може бути дробовим числом .

Поле - це множина, на якій задані операції додавання, віднімання, множення і ділення. Приклад - поле раціональних чисел (дробів). Поле Галуа - кінцеве поле (множина, що містить кінцеве кількість елементів). Зазвичай поля Галуа позначаються як $GF(p)$, де p - кількість елементів поля. Розроблено методи побудови полів необхідної розмірності (якщо це можливо). Кінцевим результатом подібних побудов зазвичай є таблиці додавання і множення, які двом елементам поля ставлять у відповідність третій елемент поля.

При реалізації алгоритмів на ПК зручно працювати з байтами. Наш алгоритм може приймати на вході N байт вихідних даних і обчислювати по ним K надлишкових байт. В одному байті може міститися 256 різних значень, тому, ми можемо створити поле $GF(2^8)$ і розраховувати надлишкові K байт, користуючись арифметикою полів Галуа. Сам підхід до кодування / декодування даних (побудова породжуючої матриці, перетворення матриці, множення матриці на стовпець) залишається таким же, як і раніше [6, 11].

Отже, в результаті ми можемо по вихідним N байтам конструювати додаткові байти, які можна використовувати при збоях. У реальних СЗД зазвичай працюють з блоками даних фіксованого розміру (в різних системах цей розмір варіюється від десятків мегабайт до гігабайтів). Цей фіксований блок розбивається на фрагментів і по ним конструюються обчислюються додаткові K фрагментів.

Обмеження арифметики раціональних чисел. Ми вже відзначали деякі труднощі, з якими доводиться стикатися при реалізації процедур кодування / декодування на конкретних апаратних платформах. Так, наприклад, числа в пам'яті ПК зазвичай представлені фіксованою кількістю байтів. Як наслідок, при множенні елементів породжуючої матриці, можна отримати переповнення розрядів. Або при оберненні породжуючої матриці в якості її елементів можуть виникнути раціональні числа, які проблематично зберігати з довільною точністю. Тому, розглянемо, як можна реалізувати дані процедури, уникаючи вищевказаних проблем.

Поля і арифметичні операції в кінцевих множинах. Перш ніж перейти безпосередньо до обговорення полів Галуа, давайте уточнимо ще раз, чого ми хочемо досягти. Ми хочемо мати можливість множити матриці на вектор, перетворювати матриці. А найголовніше, ми хочемо мати справу тільки з цілими числами і не переживати за переповнення при виконанні операцій множення і додавання.

Також, щоб уникнути плутанини, має сенс відразу розібратися з термінологією. Перш за все, що таке поле з точки зору загальної алгебри?

Отже, «поле - множина, для елементів якого визначені операції додавання, віднімання, множення і ділення (крім ділення на нуль), причому властивості цих операцій близькі до властивостей звичайних числових операцій». Поле Галуа - це довільне поле, що складаються з кінцевого числа елементів. $GF(p)$, $GF(p^n)$- стандартне позначення полів Галуа, де в дужках вказується кількість елементів поля [37 - 41].

У сучасних ПК для зберігання чисел зазвичай використовується фіксований число біт. Неважко порахувати, що всього існує 2^n різних n-бітових чисел, від 0 і до $2^n - 1$. Іншими словами, ми маємо деяке кінцеву множину чисел. На цій множині ми зараз спробуємо несуперечливим чином визначити операції множення, ділення, додавання і віднімання. Причому так, щоб результат будь-якої операції також був n - бітовим числом. У цьому випадку говорять, що множина замкнута щодо введених операцій.

Операція віднімання в системах кінцевих множин зазвичай вводиться через поняття нульового і протилежного елементів. Нульовий елемент - це такий елемент, для якого справедлива рівність: $a + 0 = a$, де a - довільний елемент множини. Елемент b є протилежним для a, якщо вірна рівність a+$b = 0$. Зазвичай протилежний елемент позначається як - a. Якщо ми хочемо з x відняти y, ми знаходимо протилежний елемент $-y$ і складаємо його с x. Відповідно, для того, щоб мати можливість віднімати довільні елементи кінцевої множини, кожен елемент цієї множини повинен мати протилежний.

Аналогічним чином йдуть справи і з операцією поділу. Вводиться поняття одиничного і зворотного елементів. Одиничний елемент 1 - це такий елемент, для якого вірна рівність $a * 1 = 0$, де a - довільний елемент множини. Елемент b (позначається як $1/a$) називається оберненим до a, якщо $a * b = 1$. Як і при відніманні, якщо ми хочемо x розділити на ненульовий y, ми повинні знайти обернений елемент $1/y$ і помножити його на x. Для того, щоб мати можливість ділити довільні елементи, кожен елемент множини (за винятком нульового) повинен володіти оберненим (таблиця 2.1).

Побудова полів Галуа GF(p). Яким чином ми можемо визначити на кінцевій множині чисел зазначені арифметичні операції, причому так, щоб множина була замкнута щодо них? Іншими словами, ми хочемо довільну кінцеву множину елементів перетворити в поле Галуа. Перше що приходить в голову, це скористатися модулярною арифметикою (таблиця 2.2). Тоді, якщо наша множина містить p елементів, то результатом множення чисел $a * b$ буде число $(a * b) mod p$. Сума чисел визначається як $(a + b) mod p$.

Складемо таблиці додавання і множення для множини, що складається з $p = 5$ елементів $\{0, 1, 2, 3, 4\}$ (таблиця 2.1, 2.2, 2.3, 2.4) [7-10].

Таблиця 2.1 – Додавання за модулем 5

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Таблиця 2.2 – Множення за модулем 5

x	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

В таблицях 2.3 і 2.4 представлені протилежні і обернені значення для кожного елемента нашої множини. Використовуючи ці таблиці, ми можемо виконувати всі необхідні нам арифметичні операції.

Таблиця 2.3 – Протилежні елементи

a	0	1	2	3	4
-a	0	4	3	2	1

Таблиця 2.4 – Обернені елементи

a	0	1	2	3	4
1/a	-	1	3	2	4

Побудова полів Галуа $GF(p^n)$. Для спрощення програмної реалізації, має сенс працювати з множинами, що містять 2^n чисел. Тоді ми зможемо оперувати напівбайтами, байтами, словами і т.д.

На перший погляд здається, яка різниця - 5 або 256 елементів у множині, беремо результат операції множення або додавання за відповідним модулем і готово. Давайте знову спробуємо скласти таблицю множення для множення, але на цей раз містить елементи - $\{0, 1, 2, 3\}$ (таблиця 2.5).

Таблиця 2.5 – Множення за модулем 4

x	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Якщо уважно подивитися на отриману таблицю, можна виявити в ній один серйозний недолік. Зверніть увагу на рядок для елемента 2. У цьому рядку немає одиничного елементу. Це означає, що ми не зможемо ділити інші елементи на 2, оскільки для нього не існує зворотного. Відповідно, необхідно будувати таблиць додавання і множення будь-яким іншим способом.

Отже, ми з'ясували, що якщо множина містить об'єкти $4 = 2^2$, то модульна арифметика не дозволяє задавати несуперечливі операції множення. Насправді, подібні проблеми виникнуть для будь-якої множини, кількість елементів якої не є простим числом.

Додавання в $GF(p^n)$. Будемо вважати, що множина, на якій ми хочемо визначити арифметичні операції містить $2^т$ елементів. Це, власне кажучи, числа. Будь-яке число з цієї множини можна представити у вигляді розкладання [41]:

$$a = a_0 + a_1 2^1 + a_2 2^2 + \cdots + a_{n-1} 2^{n-1}, a_i \in \{0, 1\}.$$

Це звичайне двійкове подання числа. Таким чином, будь-яке число з нашої множини ми можемо також розглядати як вектор довжини n, елементи якого нулі і одиниці [12, 20, 25]:

$$a = [a_0, a_1, a_2, \dots, a_n\ ], a_i\ \in \{0, 1\}.$$

Введемо операцію складання двох будь-яких чисел через складання відповідних векторів двійкового розкладання. Причому додавання компонентів векторів ми будемо вести по модулю 2 (в загальному випадку, якщо кількість елементів p^n, по модулю p). Таким чином, результуючий вектор також буде двійковим поданням деякого числа і, як наслідок, буде належати нашій множині (множина буде замкнуто щодо операції додавання).

$$a = [a_0, a_1, a_2, \dots, a_n\ ], a_i\ \in \{0, 1\};$$

$$b = [b_0, b_1, b_2, \dots, b_n\ ], b_i\ \in \{0, 1\};$$
$$c = a + b = [(a_0 + b_0) mod\ 2, \dots, (a_{n-1} + b_{n-1}) mod 2\ ].$$

Нескладно помітити, що введена нами операція додавання еквівалентна бітова операції «виключне АБО» (XOR). Що дуже добре, оскільки сучасні процесори можуть виконувати дану операцію надзвичайною швидко. Очевидно, що для довільного числа a. Це означає, що протилежним елементом до a є саме a. Таким чином, в поле $GF(2^n)$ додавання збігається з відніманням.

Множення в $GF(p^n)$. Залишилося ввести на заданій множині операцію множення. Представимо, що будь-яке число a sз нашої множини - це многочлен такого вигляду [42 - 44]:

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots +\ a_{n-1} x^{n-1}, a_i \in \{0, 1\}.$$

Тут коефіцієнти многочлена беруться з двійкового розкладу числа. При такому підході, наприклад, числу $100 = 01100100b$ буде поставлений у відповідність многочлен $x^6 + x^5 + x^2$.

Операцію множення двох чисел ми можемо визначити через множення многочленів, які відповідають даним числам:

$$a \times b = (a_0 + a_1 x + a_2\ x^2 + \cdots +\ a_{n-1} x^{n-1})\ \times$$
$$\times\ (b_0 + b_1 x + b_2\ x^2 + \cdots +\ b_{n-1} x^{n-1}.$$

Але тепер, очевидно, може виникнути інша проблема. При перемножуванні двох многочленів ступеня $n - 1$, може вийти многочлен більш високого ступеня, який не буде відповідати жодному з чисел нашої множини. Це вирішується таким чином. Вибирається неприводний многочлен ступеня n. Грубо кажучи, це такий многочлен, який не можна представити у вигляді добутку інших многочленів. Після цього, використовуючи алгоритм ділення

багаточленів стовпчиком, ми ділимо результат множення на непривідний многочлен. Ще раз нагадую, що при розподілі стовпчиком, операції з коефіцієнтами многочленів також виконуються по модулю 2. У результаті, отримуємо многочлен ступеня, не вище ніж $n-1$, який ми і приймаємо за результат добутку двох чисел. Ось приклад виконання множення двох чисел над полем $GF(2^8)$ з використанням непривідного многочлена $x^8+x^4+x^3+x+1$. Даний многочлен використовується в алгоритмі шифрування AES / Rijndael.

Ми розглянули поле $GF(2^n)$. Довільні поля $GF(p^n)$ будуються аналогічним чином, тільки замість двійкового представлення числа використовується р-ічне і всі обчислення проводяться по модулю р.

Ізоморфізм полів і поля Галуа довільного розміру. Перекладаючи з «математичного» мови, ізоморфізм означає, що відмінності тільки в позначенні елементів полів і одне можна звести до іншого, вибравши відповідно правило відображення.

Кінцеві множини елементів, які ми розглядали дотепер, були двох типів. Перший тип множин мав число елементів, рівне деякому простому числу p. На таких множинах нам вдалося задати таблиці множення і додавання, використовуючи лише модулярну арифметику. До другого типу відносяться множини, що мають p^n, p - просте, $n-1$ число елементів. На таких множинах можливо задати множення / додавання використовуючи схему з многочленами. Чи можна поставити схожим чином множення / додавання для множин довільного розміру, які наприклад містять 6 елементів? Відповідь - ні, це неможливо.

Власне кажучи, кінцеві поля і називаються полями Галуа, тому, що Галуа відкрив їх наступні важливі властивості [21-24]:

1) число елементів кінцевого поля завжди має вигляд p^n,

де p, просте число, а n будь-яке натуральне.

2) всі поля розміру p^n ізоморфні.

2.2 Обґрунтування вибору корегуючих кодів в системах зберігання інформації

Якщо провести аналогію, подібний спосіб реплікації даних - це RAID 1, простий, надійний і не дуже ефективний з точки зору витрачання місця. Ми ж хочемо створити щось аналогічне RAID 5 або RAID 6. Знову ж, йдемо від простого до складного: беремо три зони доступності, будь-яким чином розбиваємо на блоки наші дані, в зону доступності 1 записуємо парні блоки, в другу - непарні, а в третю - результат побайтового XOR між блоками. Для виявлення помилок вважаємо по кожному блоку контрольну суму, яка нехтує мала в порівнянні з розміром блоку. Відновлення даних елементарно: якщо a ^ b = c, то b = a ^ c. Коефіцієнт надмірності при такому підході - 1,5. При втраті будь-якого блоку буде потрібно прочитати два інших, причому з різних зон доступності. Відновлення можливо при втраті не більше ніж одного диска, що набагато гірше, ніж в разі трьох реплік, і дорівнює двом репліками. Ось як вважається результат XOR для рядка «Hello, habrahabr» (цифри знизу - десяткове подання байти):

Рисунок 2.6 – Обчислення контрольної суми

Тут варто ввести поняття Страйп. Страйп - це N блоків, які йдуть підряд, причому початок першого Страйп збігається з початком потоку даних, N

залежить від обраної схеми кодування, і у випадку з RAID 1 N = 2. Для ефективного застосування кодів надмірності потрібно все файли об'єднати в один безперервний потік байтів, і вже його розбивати на страйпи. Поруч слід зберегти розмітку, в якому Стайп і з якого байта починається кожен файл, а також його розмір. Якщо довжина потоку даних не кратна розміру Страйп, то треба решту заповнити нулями. Схематично це можна зобразити так (рисунок 2.7).:

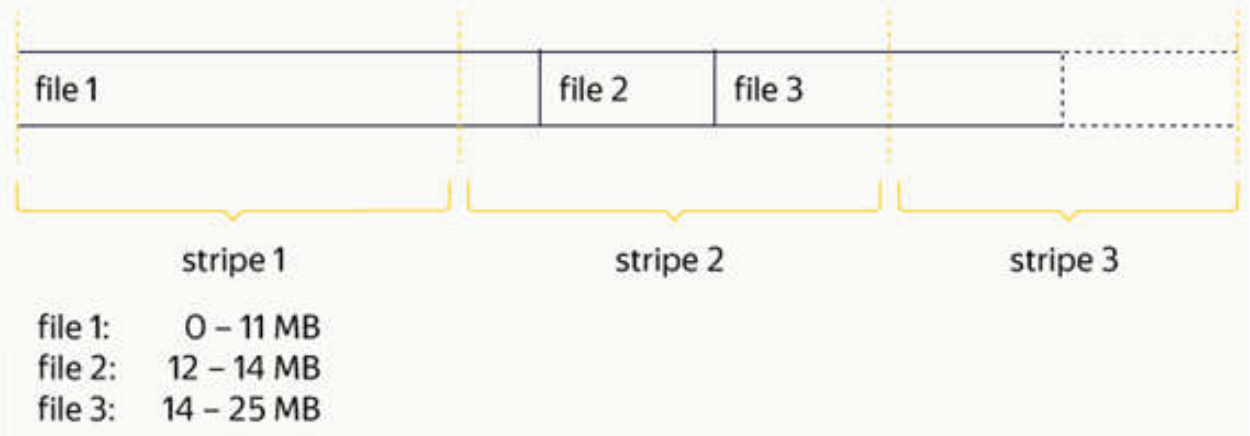


Рисунок 2.7 – Розділення файлу

При виборі розміру Страйп можна скористатися такими міркуваннями:

– чим більше розмір блоку, тим більше файлів поміститься в один блок і тим менше буде читань відразу з двох зон доступності,

– чим більше розмір блоку, тим більше даних потрібно перетягнути по мережі в разі тимчасової недоступності блоку або його втрати.

Таким чином, потрібно вважати ймовірності цих подій. У нас вийшло, що оптимальний розмір - дві медіани розміру файлу. Крім того, бажано так пересортувати файли, щоб кордони блоків припадали на більші файли, які в один блок все одно б не помістилися. Це теж призведе до зниження навантаження на мережу. А щоб спростити код, функція зберігання блоків парності закріплюється за однією із зон доступності.

Коди Ріда-Соломона. Але що робити, якщо хочеться більшої надійності? Правильно - використовувати більш потужні коди надмірності. Зараз один з

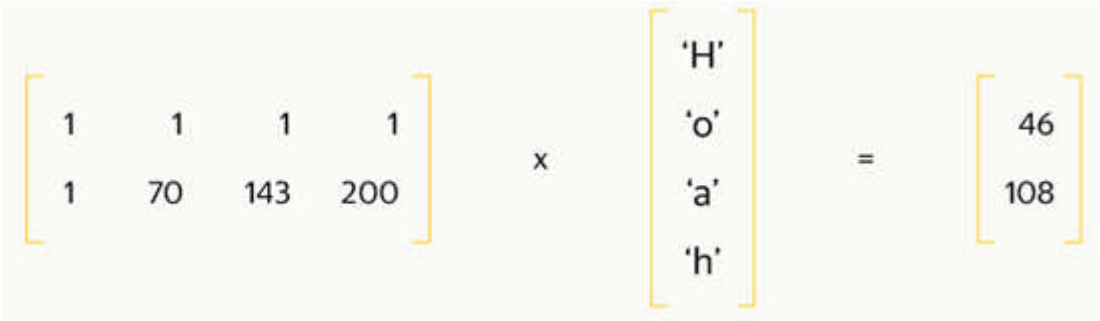
найпоширеніших кодів - це код Ріда-Соломона. Він використовується при запису DVD-дисків, в цифровому ТБ (DVB-T), в QR-кодах, а також - в RAID 6. Ми тут не будемо розповідати про математику полів Галуа - вас чекає виключно інженерний підхід. Для всіх обчислень ми задіємо бібліотеку jerasure, у якій непроста доля, але яка працює дуже швидко і володіє всіма потрібними нам функціями.

Перше, що варто відзначити: для ефективного результату потрібно працювати в полях 2 ^ 8, 2 ^ 16, 2 ^ 32, тобто в машинних словах. Далі ми для простоти будемо використовувати поле 2 ^ 8 і працювати з байтами. Щоб приклад був більш конкретним, спробуємо досягти коефіцієнта реплікації 1,5, але з двома блоками парності. Для цього буде потрібно розбити дані на страйпи по 4 блоки кожен, і згенерувати 2 блоки парності. Якщо взяти з кожного блоку даних перший байт, то можна скласти вектор розмірності 4 і, аналогічно, вектор розмірності 2 для блоків парності. Щоб з вектора розмірності 4 отримати вектор розмірності 2, його потрібно помножити на матрицю кодування розміром 2x4, причому множити треба за правилами роботи в полях Галуа, якщо дивитися з інженерної точки зору. Матриця, яка нам потрібна, називається матрицею Вандермонда. Для обраного поля така матриця гарантує властивість, аналогічне відсутності лінійних комбінацій в звичайній алгебрі. При відновленні даних воно теж зіграє важливу роль.

Візьмемо один страйп даних «Hello, habrahabr». Він дуже вдало розбивається на 4 блоки по 4 байта в кожному, причому один байт відповідає одному слову кодування (рисунок 2.8).

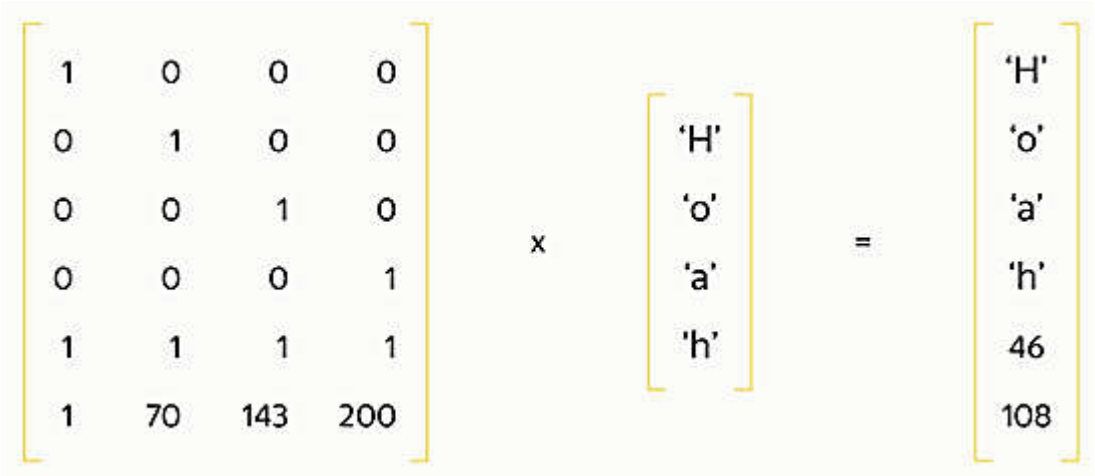
Рисунок 2.8 – Розділення повідомлення на блоки

Отже, в результаті множення отримаємо:

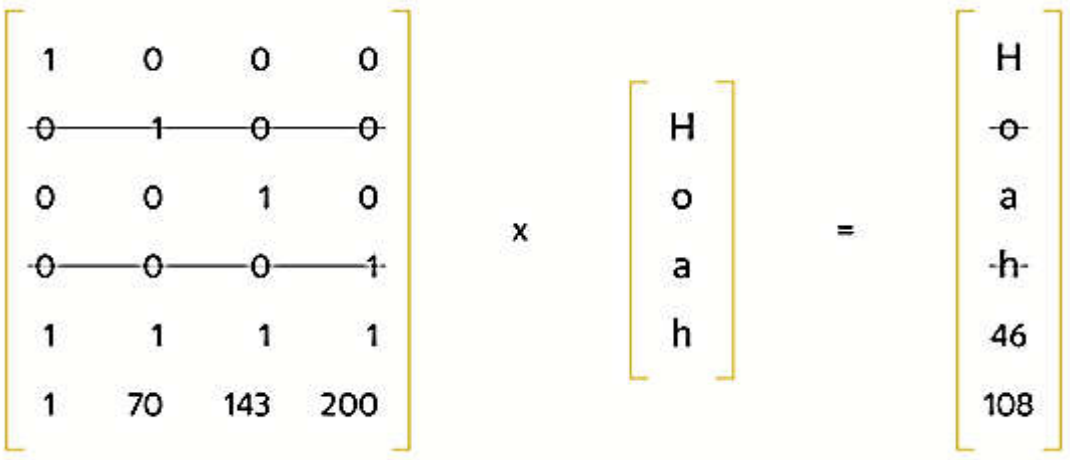


Порахуємо блоки парності подібним чином, слово за словом (в нашому випадку - байт за байтом).

Якщо додати одиничну матрицю над матрицею кодування, то в вихідному векторі з'являться вихідні дані:



Припустимо, що ми втратили блок номер 2 і блок номер 4. Викреслимо відповідні рядки з матриці і з правого вектора (рисунок 2.9):



Потім звернемо вийшла квадратну матрицю і домножимо на неї обидві частини рівності:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 143 & 142 & 141 & 2 \\ 0 & 1 & 1 & 0 \\ 142 & 143 & 140 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 70 & 143 & 200 \end{bmatrix} = \begin{bmatrix} \text{'H'} \\ \text{'o'} \\ \text{'a'} \\ \text{'h'} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 143 & 142 & 141 & 2 \\ 0 & 1 & 1 & 0 \\ 142 & 143 & 140 & 2 \end{bmatrix} \times \begin{bmatrix} \text{'H'} \\ \text{'o'} \\ 46 \\ 108 \end{bmatrix}$$

$$\begin{bmatrix} \text{'H'} \\ \text{'o'} \\ \text{'a'} \\ \text{'h'} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 143 & 142 & 141 & 2 \\ 0 & 1 & 1 & 0 \\ 142 & 143 & 140 & 2 \end{bmatrix} \times \begin{bmatrix} \text{'H'} \\ \text{'o'} \\ 46 \\ 108 \end{bmatrix}$$

Виходить, що для отримання вихідних даних потрібно всього лише провести таку ж операцію множення, як і при кодуванні! Якщо загубився всього один блок, то з матриці кодування, щоб вона стала квадратної, потрібно викреслити одну з рядків, відповідних блокам парності. Зверніть увагу, що перший рядок складається з одиниць і має особливу магію: підрахунок еквівалентний вирахування XOR між усіма елементами і виконується в кілька разів швидше, ніж підрахунок будь-якого іншого рядка. Тому викидати цей рядок не варто.

Цей метод називається локальні коди відновлення (Local Reconstruction Codes, LRC). Якщо розбити всі блоки даних на кілька груп (наприклад на дві

групи), то можна кодувати блоки парності таким чином, щоб локалізувати виправлення помилок усередині груп. Для колишнього коефіцієнта реплікації 1,5 схема виглядає так: розбиваємо страйп на дві групи по чотири блоки, для кожної групи буде свій локальний блок парності плюс два глобальних блоки парності. Ця схема дозволяє виправити будь-які три помилки і близько 96% ситуацій з чотирма помилками. До 4%, які залишилися відносяться випадки, коли всі чотири помилки доводяться на одну групу, куди входять 4 блоки даних і локальний блок парності (рисунок 2.10).

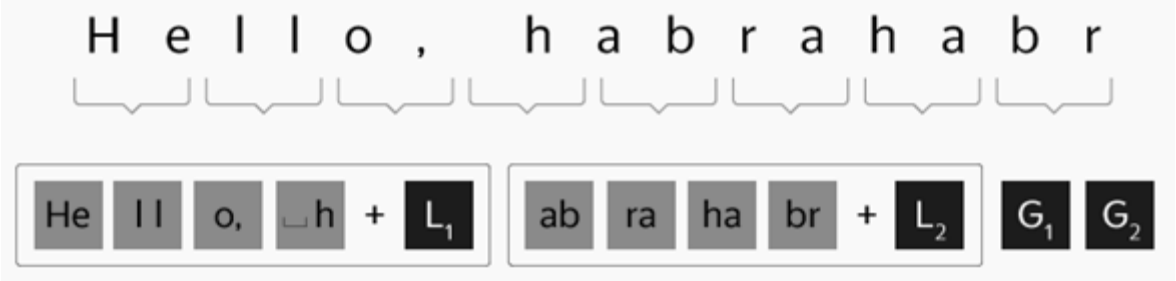


Рисунок 2.10 – Приклад кодування

Знову застосувавши підхід «від простого до складного», ми поділили рядок одиниць в матриці кодування на дві наступним чином:

1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1
1	55	39	73	84	181	225	217
1	172	70	235	143	34	200	101

На жаль, на цей раз наш підхід дав збій. Спочатку все було добре, але коли ми написали тест, який комбінаторно перебирає всі варіанти теоретично відновлюваних помилок, то виявили, що частину з них матриця відновити не дозволяє. Матрицю потрібно складати трохи по іншому, jerasure дозволяє нам з легкістю це зробити:

```
for row in range(4):
    for column in range(8):
        k = 8
        index = row * k + column
        is_first_half = column < k / 2
    if row == 0:
        matrix[index] = 1 if is_first_half else 0
    elif row == 1:
        matrix[index] = 0 if is_first_half else 1
    elif row == 2:
        shift = 1 if is_first_half else 2 ** (k / 2)
        relative_column = column if is_first_half else (column - k / 2)
        matrix[index] = shift * (1 + relative_column)
    else:
        prev = array[index - k]
        matrix[index] = libjerasure.galois_single_multiply(prev, prev, 8)
```

З такою матрицею тест проходить успішно:

1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1
1	2	3	4	16	32	48	64
1	4	5	16	29	116	105	205

Отже, ми сформували групи. Система повинна залишатися працездатною при падінні цілої зони доступності. З використанням LRC цього можна домогтися, якщо розташувати блоки по зонам наступним чином (рисунок 2.11).

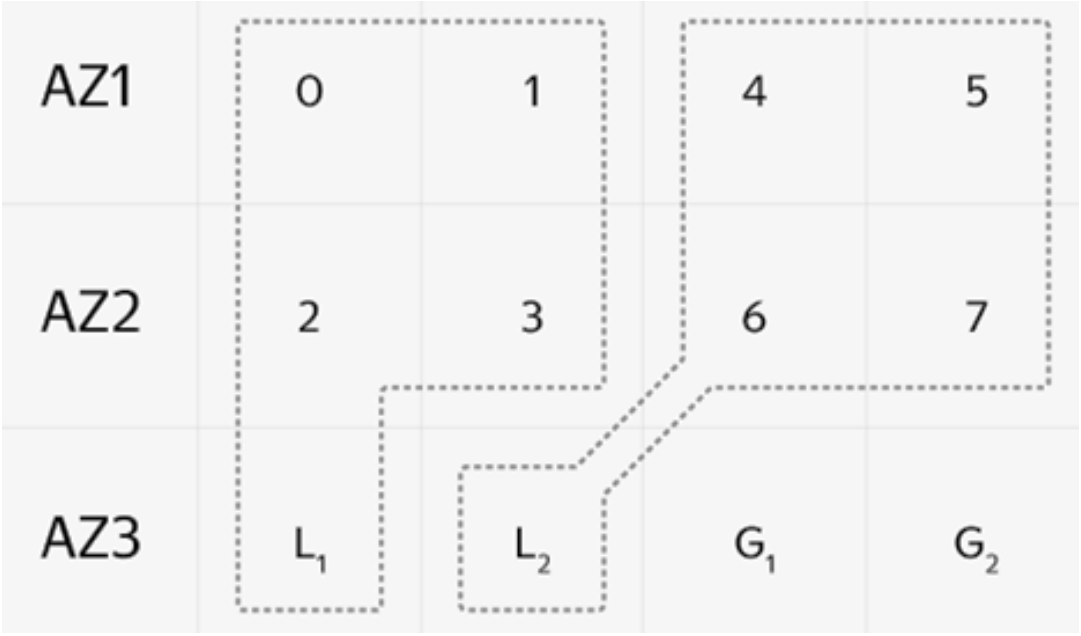


Рисунок 2.11– Розташувати блоків по зонах доступності

Тут видно, що який б рядок ми не закреслили, в кожній групі локальності виявиться не більше трьох помилок, а значить, дані можна буде прочитати. Якщо ж зламається тільки один диск, на якому знаходиться один блок, то дані можна буде прочитати, запросивши з іншої зони доступності всього лише один додатковий блок. Міркування з приводу розміру блоку - приблизно такі ж, як і в схемі з XOR, з одним винятком: якщо вважати, що читання виявиться занадто дорогим тільки між зонами доступності, то розмір блоку можна зробити в чотири рази менше, так як вийде безперервна послідовність в межах однієї зони.

Отже, запрограмувати обчислення надлишкових кодів завдання не складне. Ми розглянули наступні варіанти:

– репліки - дають кратну надлишковість, легко реалізувати, працюють як на блочному, так і на файловому рівні.

– XOR, надлишковість - 1,5, можна реалізувати як на блочному, так і на файловому рівні, вимагає наявності трьох зон доступності.

Коди Ріда-Соломона - дозволяють гнучко вибирати ступінь надлишквості, але для відновлення одного блоку даних потрібно прочитати

стільки ж блоків, скільки містить один страйп. Добре працюють тільки на блочному рівні.

LRC - схожі на коди Ріда-Соломона, але при одиничних збоях дозволяють читати менше даних, хоча і забезпечують відновлення даних в меншому відсотку випадків.

3 РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ СИСТЕМИ ЗБЕРІГАННЯ ДАНИХ

3.1 Структура системи зберігання даних з використанням коригуючих кодів

Перш ніж розробляти пристрій розподіленого зберігання, треба сказати про коди Ріда - Соломона (RS), які запропоновані в 1960 році співробітниками Лінкольнівскої лабораторії Массачу́сетського технологічного інституту Ірвіном Рідом і Густавом Соломоном для використання в системах далекого космічного зв'язку. У 80-ті роки почалося широке комерційне застосування RS - спочатку в аудіодисках для забезпечення повного збереження запису і чистоти відтворення, незважаючи на зовнішні пошкодження, а пізніше RS коди впровадили в DVD і Blu-ray. Крім того, вони знайшли своє місце в системах передачі даних DSL і WiMAX, в стрічкових запам'ятовуючих пристроях, в контролерах оперативної пам'яті і багатьох інших застосуваннях. Зазвичай ці коди використовуються в тих випадках, коли є необхідність за рахунок надлтшковості забезпечити надійну передачу даних по ненадійних каналах.

Можна сказати, що коди Ріда - Соломона є логічним розвитком відомих кодів Хемінга, але вони відрізняються здатністю виправляти більшу кількість помилок, які виникають одночасно. Математична теорія циклічних кодів з виправленням помилок RS досить складна, але основні її положення можна викласти, не вдаючись до математики. Суть в тому, що до послідовності з n символів у вихідному повідомленні додається певна кількість (m) коригувальних символів, які змішуються з алгоритмами RS-кодування. Закладена в повідомлення надлишковість дозволяє виявляти не одну, як в коді Хемінга, а m помилок і виправляти їх до половини. Число коригувальних символів може бути менше або дорівнювати числу вихідних, відношення (n / m) називають коефіцієнтом резервування (Redundancy Rate). На рисунку 3.1 байт, який передається містить п’ять змістовних бітів, до яких додано три контрольних. Кожен з байтів передається по одному з восьми можливих

маршрутів, і якщо при передачі по трьом будь-яких каналах відбувається збій, то решти п'яти досить для відновлення вихідних даних.

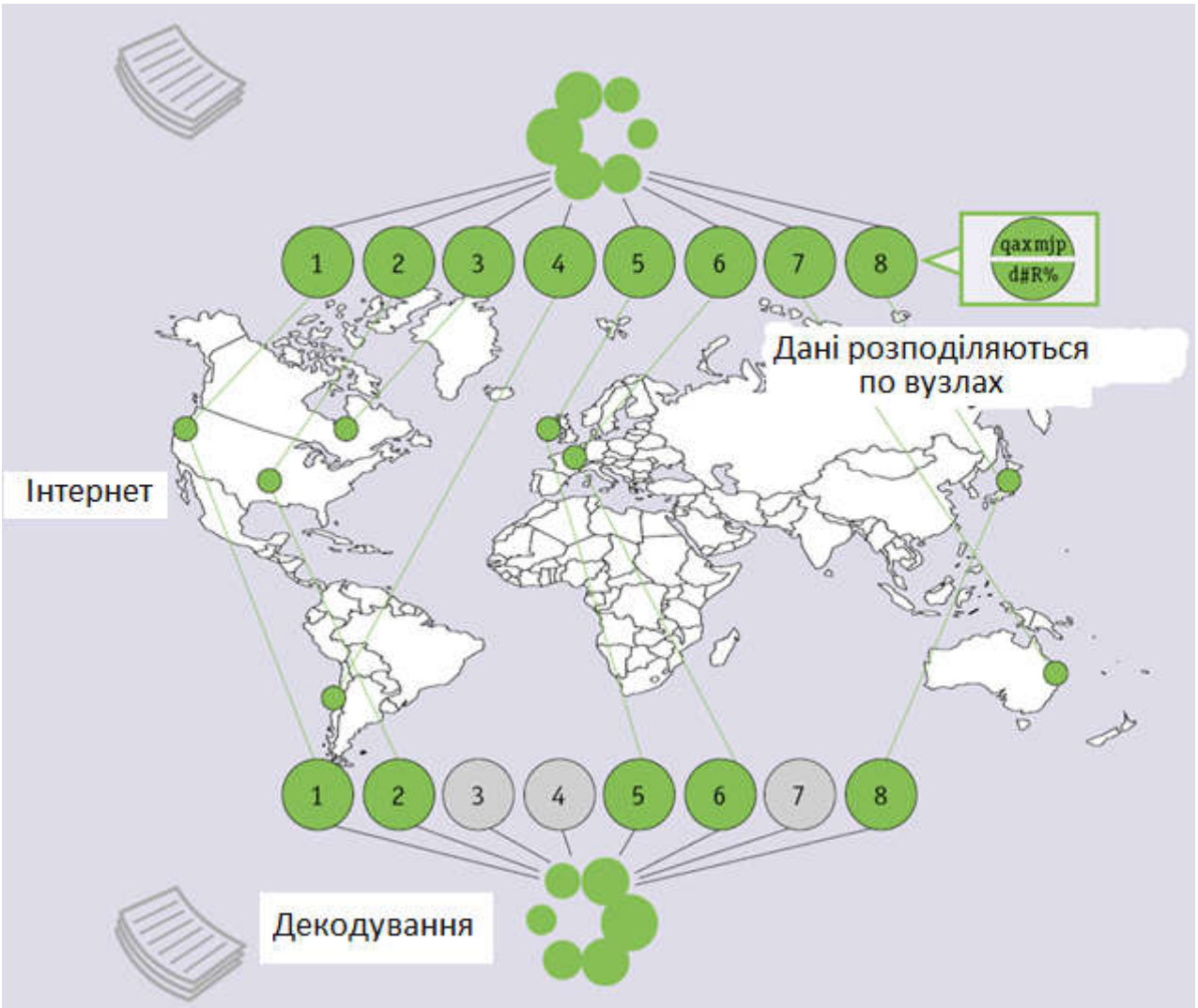


Рисунок 3.1 – Принцип розподіленого зберігання даних

Задача в тому щоб використати коригувальний потенціал кодів RS в системах зберігання даних, що дозволяє відмовитися від класичних технологій, що використовуються для забезпечення схоронності даних. При записі зберігаються дані розподілені на «частини» і засобами IDA розподіляються по географічно рознесених дисках, а читання проводиться тими ж засобами IDA шляхом збору даних з цих пунктів зберігання. Якщо якась частина не відповість або надішле дані з помилкою, то ті, що залишилися має вистачити для відновлення даних. Можливість до самоуправління і самовідновлення критично важливі, коли справа стосується великих обсягів даних, а класичні

методи резервування перестають працювати, наприклад при створенні систем резервного копіювання петабайтного і більше обсягу.

Всі відомі методи, націлені на збереження даних, засновані в кінцевому рахунку на інженерних підходах до дублювання і віддзеркалення, і, як наслідок, вимагають відведення для них помітних обсягів на дисках або стрічках, складних і дорогих процедур з переміщення на пристроях в залежності від їх затребуваності, і при цьому дані зберігаються «однією частиною», що спрощує їх крадіжку. У разі розподіленого сховища – дані розподілені - немає майданчика, де б зберігався весь масив. Важко уявити собі подію, яка б торкнулася б відразу всіх пунктів зберігання, розподілені по всій земній кулі. А оскільки ми маємо справу з «нарізкою», крадіжка однієї з «частинок» позбавлена сенсу - дані природним чином захищені.

Розподілене сховище складається з шлюзів (Gateway) і внутрішніх серверів (Internal Server). Користувач має доступ лише до одного з найближчих до нього шлюзів, а шлюзи через Інтернет спілкуються з серверами. Логіка зберігання IDA реалізується шлюзами, які здійснюють оброблення даних на «частини», розподіл до віддалених серверів зберігання і виконання зворотної операції збірки даних в єдине ціле з «частин» з корекцією в разі виявлення помилок. Шлюзи перевіряють також сигнатури користувачів і серверів. Комунікація між клієнтами і системою підтримується по протоколу SSL, а шлюзи звертаються до серверів по інтерфейсу Common Gateway Interface. Інформація на тракті від клієнта до сервера шифрується за алгоритмом RSA з використанням цифрових сертифікатів.

Серед характеристик систем зберігання даних, які є найважливішими з точки зору кінцевого користувача необхідно виділити наступні: швидкість операцій введення / виводу, вартість використання системи, рівень енергоспоживання та інші. Але всі ці характеристики будуть мати сенс тільки в тому випадку, якщо система забезпечує необхідний користувачеві рівень надійності збереження даних. Як результат, для всіх розробників подібних

систем, питання забезпечення надійності зберігання даних виходять на перший план.

Довгий час головним гравцем в сфері безпечного зберігання даних залишалася технологія RAID. Подібні класичні рішення, поряд з відомими плюсами, володіють і певними обмеженнями. До них можна віднести:

– відносно низький рівень захисту (наприклад, RAID 6 підтримує вихід з ладу максимум двох дисків);

– необхідність тримати запасні диски;

– неможливість контролювати розподіл «parity» блоків (це може бути важливо для певних типів дисків);

– складність підтримки «гетерогенних» дисків.

У зв'язку з цим, останнім часом, набирають популярність технології, засновані на різних надлишкових кодах (в англомовній літературі за подібними технологіями закріпилася назва - erasure coding), які надають набагато більше можливостей у порівнянні з класичним RAID.

Після проведеного аналізу коригуючих кодів для реалізації системи зберігання даних вирішили зупинитися на кодах Ріда-Соломона. Існують готові бібліотеки (наприклад, ISA-L від Intel або Jerasure авторства James S. Plank). У них реалізовані відповідні процедури кодування / декодування. Але з огляду на те, що апаратна платформа буде побудована на базі архітектури ПЛІС, а вся логіка системи реалізована як модуль ядра ОС Linux, - було прийнято рішення робити свою реалізацію, оптимізовану під особливості нашої системи.

При реалізації алгоритмів на ЕОМ зручно працювати з байтами. Наш алгоритм може приймати на вході байт вихідних даних і обчислювати по ньому перевірочний байт. В одному байті може міститися 256 різних значень, тому, ми можемо створити поле і розраховувати надлишковий байт, користуючись арифметикою полів Галуа. Сам підхід до кодування / декодування даних (побудова породжує матриці, звернення матриці, множення матриці на стовпець) залишається таким же, як і раніше.

У реальних системах зберігання даних зазвичай працюють з блоками даних фіксованого розміру (в різних системах цей розмір варіюється від десятків мегабайт до гігабайтів). Цей фіксований блок розбивається на фрагменти і по ним формуються додаткові фрагменти (рисунок 3.2) [35].

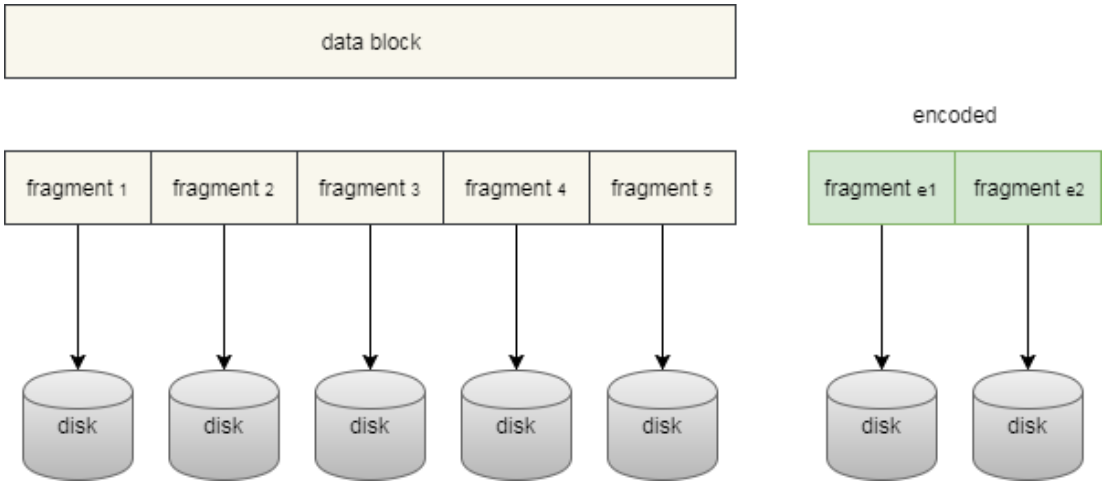


Рисунок 3.2 – Структура системи зберігання даних з використанням корегуючих кодів

Процес обчислення фрагментів відбувається наступним чином. Беруть по одному байту з кожного з вихідних фрагментів зі зміщення 0. За цими даними генерується K додаткових байтів, кожен з який йде до відповідних додаткові фрагментів зі зміщення 0. Цей процес повторюється для зміщення - 1, 2, 3, ... Після того, як процедура кодування закінчена, фрагменти зберігаються на різні диски.

При виході з ладу одного або декількох дисків, відповідні втрачені фрагменти реконструюються і зберігаються на інших дисках.

Базова задача відновлення втраченого. Припустимо, що є два довільних цілих числа A і B. Необхідно розробити алгоритм, який двом цим числам поставить у відповідність третє число R, причому таке, що по ньому буде можливо однозначно відновити будь-яке з двох вихідних чисел при втраті одного з них. Іншими словами, маючи пару (A і B) або (B і R), можливо однозначно відновити B або A відповідно. Існує багато варіантів реалізації подібного алгоритму, але, ймовірно, найпростішим є той, який до двох

вихідних чисел застосовує бітову операцію «Виключне« АБО »(XOR)», тобто $R = A\ xor\ B$. Тоді для відновлення можна скористатися рівністю $A = RxorB$(для відновлення B, $B = R\ xor\ A$).

Розглянемо складніше завдання. Припустимо, що вихідних чисел не два, а три - $A, B\ i\ C$. Як і раніше, потрібен алгоритм, що дозволяє впоратися з втратою одного з вихідних чисел. Легко помітити, що ніщо не заважає знову використовувати бітову операцію «Виключне« АБО», $R = A\ xor\ B\ xor\ C$. Для відновлення Aможна скористатися рівністю $A = R\ xor\ B\ xor\ C$.

Розглянемо ще один варіант. Як і раніше, є три числа, A, B і C, а ось втратити, в цей раз, ми можемо не одне, а будь-які два з них. Очевидно, що просто використовувати операцію «Виключне« АБО» вже не вийде. Одним, з можливих шляхів вирішення даної задачі, є застосування кодів Ріда-Соломона.

Технології відновлення даних неспрості називаються методами надлишкового кодування. За вихідними даними обчислюються деякі «надлишкові», які потім дозволяють відновити втрачені. Не вдаючись в подробиці зауважимо, що емпіричне правило таке - чим більше даних може бути втрачено, тим більше «надлишкових» необхідно мати. У нашому випадку для відновлення двох чисел, нам доведеться за певним алгоритмом сконструювати два «надлишкових». У загальному випадку, якщо потрібно підтримувати втрату K чисел, необхідно відповідно мати K надлишкових. Для вирішення даної задачі використаємо «алгебраїчний» підхід.

3.2 Алгоритм роботи системи зберігання даних

Основним завданням більшості сучасних систем зберігання даних є одночасне надання ресурсів зберігання декількох клієнтських станцій (ініціаторам) [13-19].

Слід розділяти завдання, що вимагають ресурсів зберігання, на критичні і некритичні для бізнесу компанії. Неможливість виконання критичних для бізнесу завдань в зв'язку з тим, що всі необхідні ресурси були захоплені додатками, що виконують некритичні завдання, може призвести до серйозних фінансових втрат.

Досить часто надання рівня обслуговування за запитами різних ініціаторів формується вручну системним адміністратором.

Адміністратор може надати пріоритет одному або декільком ініціаторам, тоді запитам від них буде надана гарантована пропускна здатність. Однак такий спосіб управління в системах зберігання даних не може забезпечити рівень обслуговування з оптимальною продуктивністю і надійністю.

Технологія автоматичного подання пріоритету QoSmic, заснована на ідентифікації працюючих додатків в системі зберігання даних (СЗД) «на льоту». За допомогою алгоритму QoSmic розпізнаються критично важливі бізнес-додатки, і саме їм виставляється найвищий пріоритет. Пріоритет автоматично знімається, коли критично важливий додаток перестає працювати. Даний алгоритм може бути включений або вимкнений за бажанням клієнта. Система зберігання даних з вбудованою технологією QoSmic представлена на рисунку 3.3.

Опишемо докладніше дану технологію. Алгоритм ідентифікації додатків складається з двох модулів: навчання та розпізнавання.

Навчання проходить в офлайн-режимі. На основі I/O - запитів від запущеного додатку визначаються сигнатури запитів на читання і запис за певним алгоритмом [33], далі з даних сигнатур формується профіль додатку. Профілі додатків надходять в алгоритм класифікації Random Forest [34]. На виході Random Forest видає модель (див. рисунок 3.3), який використовується в модулі розпізнавання додатків в якості провісника.

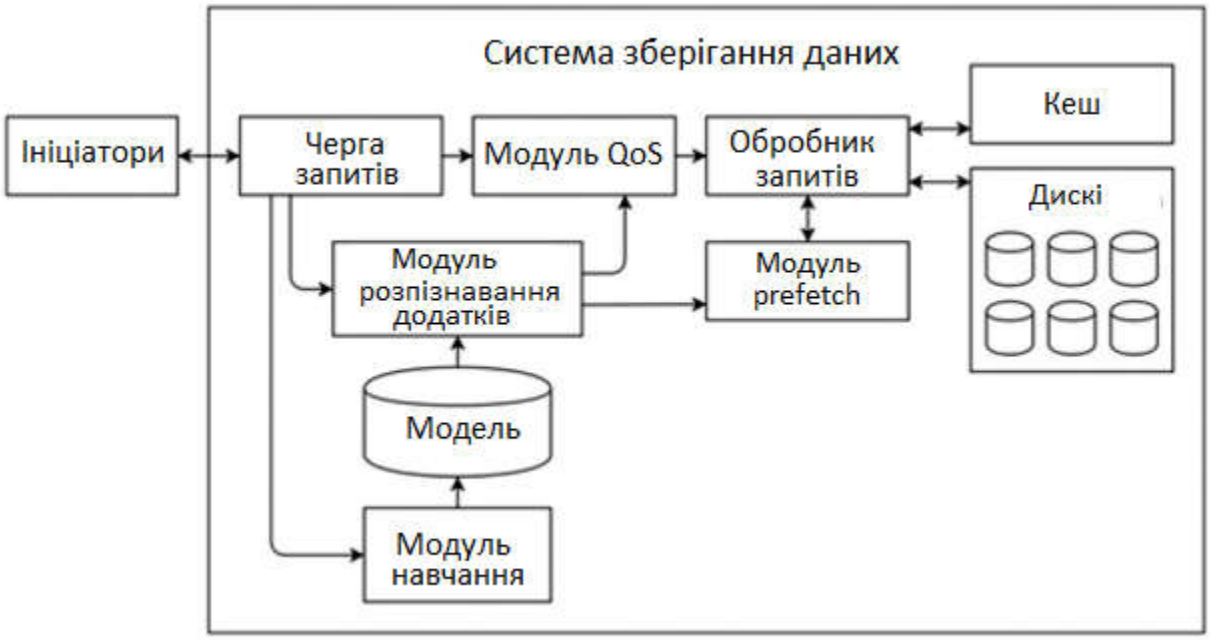


Рисунок 3.3 – Система зберігання даних з вбудованою технологією QoSmic

Механізм розпізнавання працює в реальному режимі. На основі I/O-запитів від ініціатора, які поступають в модуль розпізнавання додатків, будуються I/O-сигнатури, що є вхідними даними для алгоритму розпізнавання. Далі за допомогою алгоритму Random Forest і моделі, отриманої в режимі навчання, ідентифікуються програми, що працюють на ініціатора, або видається відповідь «не вдалося визначити».

Така відповідь видається в разі, якщо на ініціатора працює невідомий додаток або часу для ідентифікації недостатньо, або інтенсивність I / O-запитів низька, що не дозволяє сформувати «надійні» I / O - сигнатури. Далі імена додатків надходять в модуль QoS, який і виставляє пріоритети тих чи інших ініціаторам.

Алгоритм точно ідентифікує додатки. Імовірність помилок першого роду (коли важливий додаток порахували неважливим) або другого роду (коли один критично важливий додаток було прийнято за інший) дуже мала і представлена в таблиці 3.1.

Таблиця 3.1 – Точність ідентифікації

	Помилки першого роду, %	Помилки другого роду, %
Apple Final Cut Pro/X	0,1	0,5
Adobe Premiere Pr	0,15	0,8
Autodesk Smoke	0,12	0,7
Antivirus Business Security	0,01	0,01
MSSQL база даних	0,005	0,01
Неважливі додатки	0,1	-

Високий рівень точності, досягнутий завдяки обраним параметрам в I / O - сигнатури, дозволяє формувати досить точний статистичний профіль додатків і, отже, з високою точністю виявляти працює критично важливий додаток і відрізняти його від низько пріоритетного.

Алгоритм розпізнавання не вимогливий до ресурсів і не впливає на продуктивність системи. При максимальному за інтенсивністю навантаженні на систему зберігання даних з малопотужним процесором і недорогими HDD споживчого класу продуктивність системи практично не змінюється, тобто кінцевий користувач не відчуває ніяких затримок в роботі системи як при включеному, так і при вимкненому режимі QoSmic (рисунок 3.4).

Ідентифікація додатків на блочному рівні в системах зберігання даних є унікальною функцією на ринку СЗД. Алгоритм має високу точність і швидкість, що дозволяє використовувати його для автоматичного виставлення пріоритету критично важливих додатків і гарантувати даним додатків потрібну пропускну здатність незалежно від навантаження з боку інших працюючих додатків. Даний алгоритм здатний відрізняти характерні патерни і всередині програми, наприклад, обробку відео 4K від відео 2K.

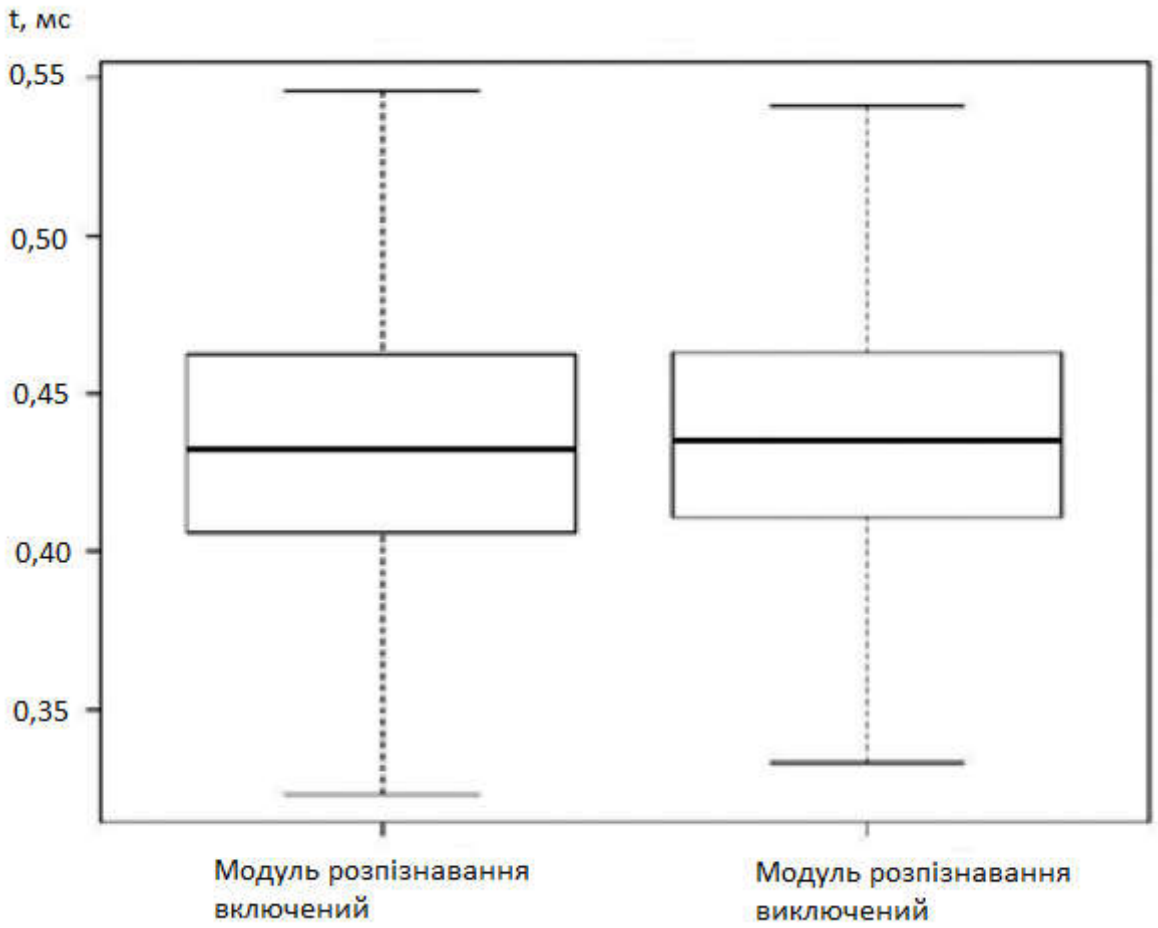


Рисунок 3.4 – Діаграма розподілу продуктивності СЗД

RAIDIX дозволяє створювати дискові масиви з різними рівнями надійності. Серед них є як стандартні масиви для систем зберігання RAID-0, RAID-5, RAID-6, так і RAID з більш високим рівнем надійності - RAID N + M. Назва RAID N + M відображає той факт, що для N дисків з даними ми можемо забезпечити M дисків з контрольними сумами:

RAID N+M в RAIDIX.

Про те, як такий підхід підвищує надійність, а також про деякі особливості реалізації кодування в такому RAID масиві розглянемо далі.

У СЗД розрізняють 2 типу втрати даних. Перший пов'язаний з виходом з ладу диска або його частини, який виявляється системами технічного контролю. В першому випадку місце втрати даних відомо, і прийнято говорити

про відмову або виході з ладу диска. Під другому випадку дані спотворюються в процесі запису, зберігання або читання таким чином, що системи апаратного контролю не виявляють збою дисків. При частковій втраті місце втрати даних невідомо, так само як невідомий і сам факт втрати. Відповідно, необхідно виявити проблему, знайти місце спотворення і відновити дані на основі надлишкової інформації. У разі, коли диск не виходить з ладу, а дані на ньому спотворені, кажуть про приховані пошкодженні даних - SDC (Silent Data Corruption). Так, недавнє дослідження, проведене на 1,5 млн HDD в базі даних NetApp протягом 2,5 років, виявило приховані пошкодження на 8,5% дисків SATA.

Різні рівні RAID можуть піддаватися будь-яким з цих пошкоджень. Чим більше пошкоджень здатний пережити RAID, тим надійніше система.

Наприклад, якщо в дисковому масиві розраховується m контрольних сум і при цьому відбулося p відмов дисків і l випадків SDC, то, при виконанні умови $p + 2l \leq m$, всі відмови і приховані пошкодження можуть бути виявлені і виправлені. вибираючи різне кількість контрольних сум, можна домогтися бажаного рівня надійності залежно від розміру сховища. В основі даного рівня RAID лежать коди Ріда-Соломона. З метою виконання кодування дані, що записуються в СЗД, розглядаються як послідовності кодових слів. Щоб визначити операції складання, множення, ділення кодових слів використовується арифметика полів Галуа. Кожне кодове слово розглядається як поліном з цього поля. Додавання двох кодових слів - це складання поліномів по модулю 2 (логічна операція XOR), а множення полягає в тому, щоб перемножити відповідні поліноми і взяти залишок від ділення на який утворює поле не приводиться многочлен.

Наприклад, якщо ми розглядаємо кодові слова як елементи поля GF (2^8), утвореного многочленом f = x8 + x4 + x3 + x2 + 1 = (100011101), тоді для двох кодових слів

$$a1 = (10001001) = x^7 + x^3 + 1,$$

$$a2 = (11110000) = x^7+x^6+x^5+x^4,$$

буде виконуватися

$$a1+a1=(01111001) = x^6+x^5+x^4+x^3+1,$$

$$a1\times a1=a1\times a1 \ (\mathrm{mod}\, f) = (10010010) = x^7+x^4+x.$$

Даний спосіб реалізації операцій дозволяє не тільки складати і множити, але і ділити байти так, щоб результат мав той же розмір, що і операнди.

Визначивши дії з кодовими словами, можна переходити до реалізації кодів Ріда-Соломона. У їх класичної реалізації при необхідності виправлення прихованих пошкоджень для розрахунку контрольних сум будуються поліноми над полем Галуа. Це такі поліноми, коефіцієнти яких і є нашими кодовими словами. Далі з цими поліномами виконуються певні дії: множення, пошук залишків від ділення та інші. В результаті виходить багаточлен, коефіцієнти якого є контрольними сумами.

Однак, це не найшвидший підхід до розрахунків, щоб підвищити швидкість кодування і декодування. Основна ідея - в переході від дій з поліномами над кінцевим полем до матричних кодування. Інакше кажучи, необхідно вирахували такі матриці (матриці кодування і декодування), множення на які дає той же результат, що і дії з поліномами над полем.

Розрахунок таких матриць досить простий, а множення на них може бути розпаралелено.

Наприклад, для n кодових слів D_0, D_1, ..., D_{n-1} розрахунок m контрольних сум C_0, C_1, ..., C_{m-1} виконується наступним чином [37]:

$$C_m = \begin{pmatrix} \tilde{W}_1(a^{N-1}) & \tilde{W}_1(a^{N-2}) & \dots & \tilde{W}_1(a^m) \\ \tilde{W}_2(a^{N-1}) & \tilde{W}_2(a^{N-2}) & \dots & \tilde{W}_2(a^m) \\ \dots & \dots & \dots & \dots \\ \tilde{W}_m(a^{N-1}) & \tilde{W}_m(a^{N-2}) & \dots & \tilde{W}_m(a^m) \end{pmatrix} D_n.$$

де a - примітивний елемент поля, N = n + m.

Дані співвідносяться так:

$$\tilde{W}_j = \frac{(x-\lambda_1)(x-\lambda_2)\dots(x-\lambda_{j-1})(x-\lambda_{j+1})\dots(x-\lambda_m)}{(\lambda_j-\lambda_1)(\lambda_j-\lambda_2)\dots(\lambda_j-\lambda_{j-1})(\lambda_j-\lambda_{j+1})\dots(\lambda_j-\lambda_m)} =$$

$$= w_{j,0} + w_{j,1}x + \dots + w_{j,m-1}x^{m-1}.$$

Для фіксованих значень N і M ця матриця може бути розрахована заздалегідь і використовуватися при кожному кодуванні. У разі виникнення відмов аналогічним чином будується матриця декодування. Помноживши її на робочі блоки, можна знайти втрачені значення.

Складніше відбувається пошук прихованих пошкоджень. Щоб визначити, чи мало місце спотворення даних, блоки D_0, D_1, ..., D_{n-1}, C_0, C_1, ..., C_{m-1} множаться на матрицю Вандермонда. Якщо серед отриманих в результаті множення значень є ненульові, значить, відбулися приховані пошкодження. Щоб знайти і виправити їх, отримані значення передаються на вхід алгоритму Берлекампа-Мессі, який дозволяє знайти місця всіх помилок.

Розроблена реалізація алгоритмів завадостійкого кодування, де не тільки використані алгоритмічні оптимізації, але і всі дії в полях Галуа реалізовані з допомогою векторних регістрів процесора. Векторизація обчислень істотно підвищує швидкість кодування і декодування. RAIDIX також використовує векторні інструкції і в алгоритмі Берлекампа-Мессі, щоб ефективно знаходити місця прихованих пошкоджень.

Таким чином, за рахунок використання матричного кодування і векторизації основних операцій в розрахунках досягнуто високої швидкості кодування і декодування навіть в разі виникнення великої кількості відмов і прихованих пошкоджень. Здатність дискового масиву переживати такі складні випадки пошкодження забезпечує високу ступінь надійності зберігання даних в системі.

3.3 Дослідження корегуючих кодів в системах зберігання даних

Відомо, що головна перевага програмної реалізації кодів - це низька вартість та простота реалізації. Але при цьому у неї є такі недоліки як низька продуктивність, завантаження додаткової роботи центрального процесора.

Апаратні реалізації коштують дорожче, ніж програмні, так як використовуються додаткові прилади для виконання операцій. При цьому вони розвантажують центральний процесор та системну шину і дозволяють збільшити швидкість роботи, за рахунок заміни послідовностей операцій на операцію, яка виконується за один такт. Більшу функціональну потужність можна досягти шляхом застосування компонентної бази - програмованих логічних інтегральних схем (ПЛІС). Таким чином, застосування ПЛІС дає велику апаратну гнучкість за рахунок їх реконфігурації, а також можливість видалення та, при необхідності, виготовлення в кремнії без надлишковості, яка наявні в ПЛІС.

Для дослідження були обрані коди за такими критеріям. Код для перевірки цілісності даних - CRC (Cyclicredundancycheck) код, який базується на поліноміальній арифметиці. Код, який призначений для виправлення одноразової помилки - код Хемінга (найпростіший лінійний код) і код для виправлення багатократних помилок - код Ріда-Соломона (недвійковий лінійний код) [1-3, 37].

Мовою опису апаратних засобів було обрано System Verilog, яка використовується для опису і верифікації апаратури, що є розширенням мови Verilog. Для дослідження апаратної реалізації даних алгоритмів використовувалася ПЛІС типу FPGA сімейства Cyclone III, розробленої фірмою Altera на 39600 логічних елементів. В якості середовища проектування було обрано Quartus II, яке підтримує дану ПЛІС, а також інструмент логічного синтезу DesignVision.

Основною перевагою апаратної реалізації завадостійких кодів на FPGA є паралельне обчислення. Апаратна реалізація алгоритмів в першу чергу потрібна для отримання максимальної швидкодії виконання операцій кодування і декодування в пристрої.

Блок складається з регістра управління, регістра контрольної суми і логіки розрахунку CRC (рисунок 3.5). За допомогою регістра управління можна: включити і вимкнути режим початкового заповнення; встановити формат записуваних даних. Одним з бітів регістру управління є CI - біт ініціалізації контрольної суми. Мультиплексор управляється сигналом FORM - поле завдання формату записуваних даних. Тригер застосовується для замикання вхідних даних. Регістр контрольної суми використовується для ініціалізації контрольної суми, прийому даних, для яких обчислюється контрольна сума і зчитування отриманого значення контрольної суми (рисунок 3.6).

Кодування і декодування кодів Ріда-Соломона є досить складним завданням. Його рішення виливається в громіздкий, заплутаний і вкрай неочевидний код, який вимагає широких знань від розробника в багатьох областях вищої математики [30].

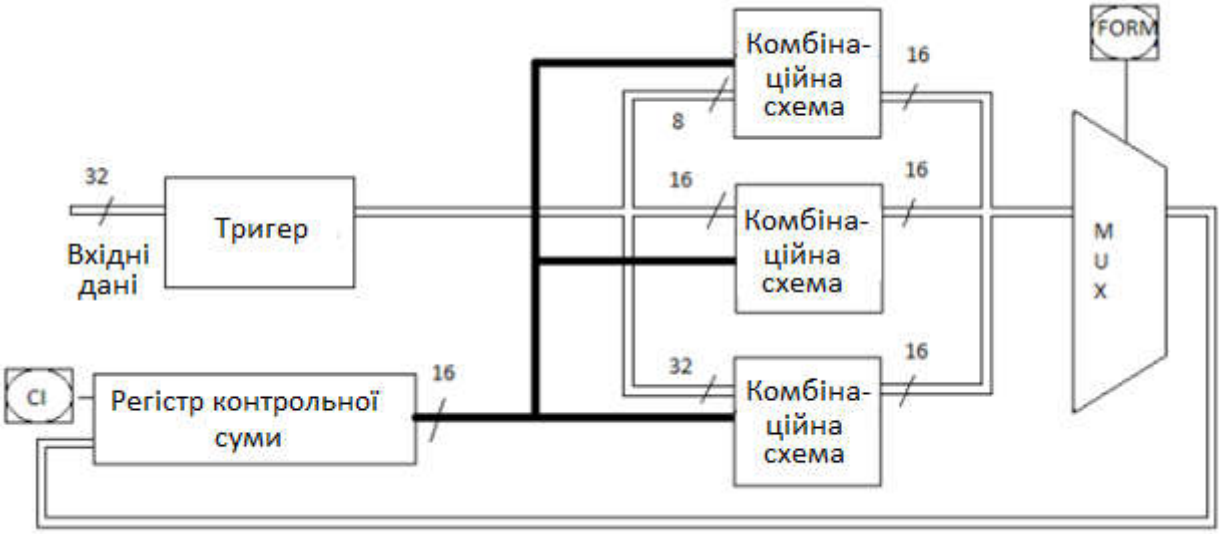


Рисунок 3.5 – Функціональна схема блоку обчислення CRC коду для одного байту, слова і подвійного слова

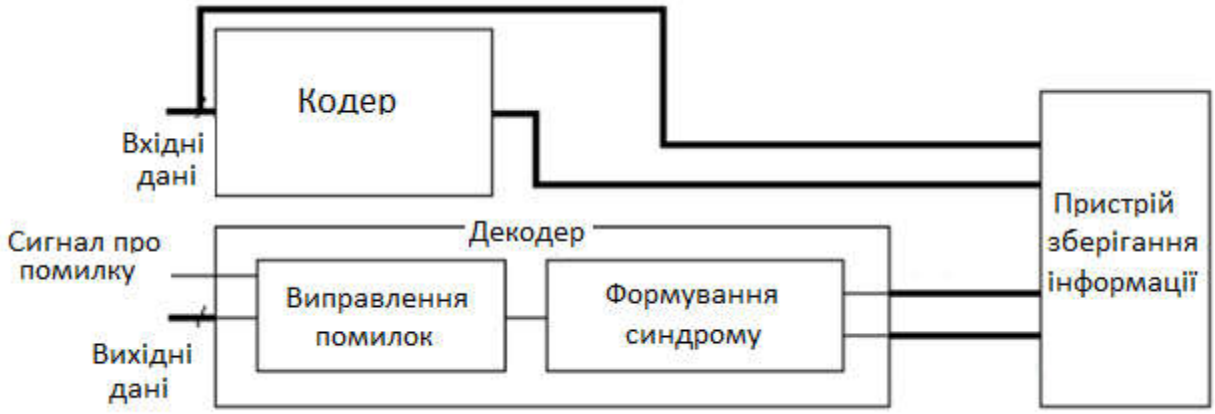


Рисунок 3.6 – Функціональна схема блоку обчислення коду Хемінга

На рисунку 3.7представлена типова схема декодування, яка отримала назву авто регресійного спектрального методу декодування. Існує багато різних схем для декодування даних з використанням коду Ріда-Соломона. Представлена на рисунку 3.7 схема декодування є універсальною. Архітектура кодера являє собою з'єднання зсувних регістрів, які об'єднані за допомогою суматорів і помножувачів, що функціонують за правилами арифметики Галуа [1].

Важливим етапом розробки є синтез логічної схеми. Логічний синтез - це процес отримання списку з'єднань логічних вентилів з абстрактної моделі поведінки логічної схеми (наприклад, на рівні реєстрових передач, RTL – Register Transfer Level) [2].

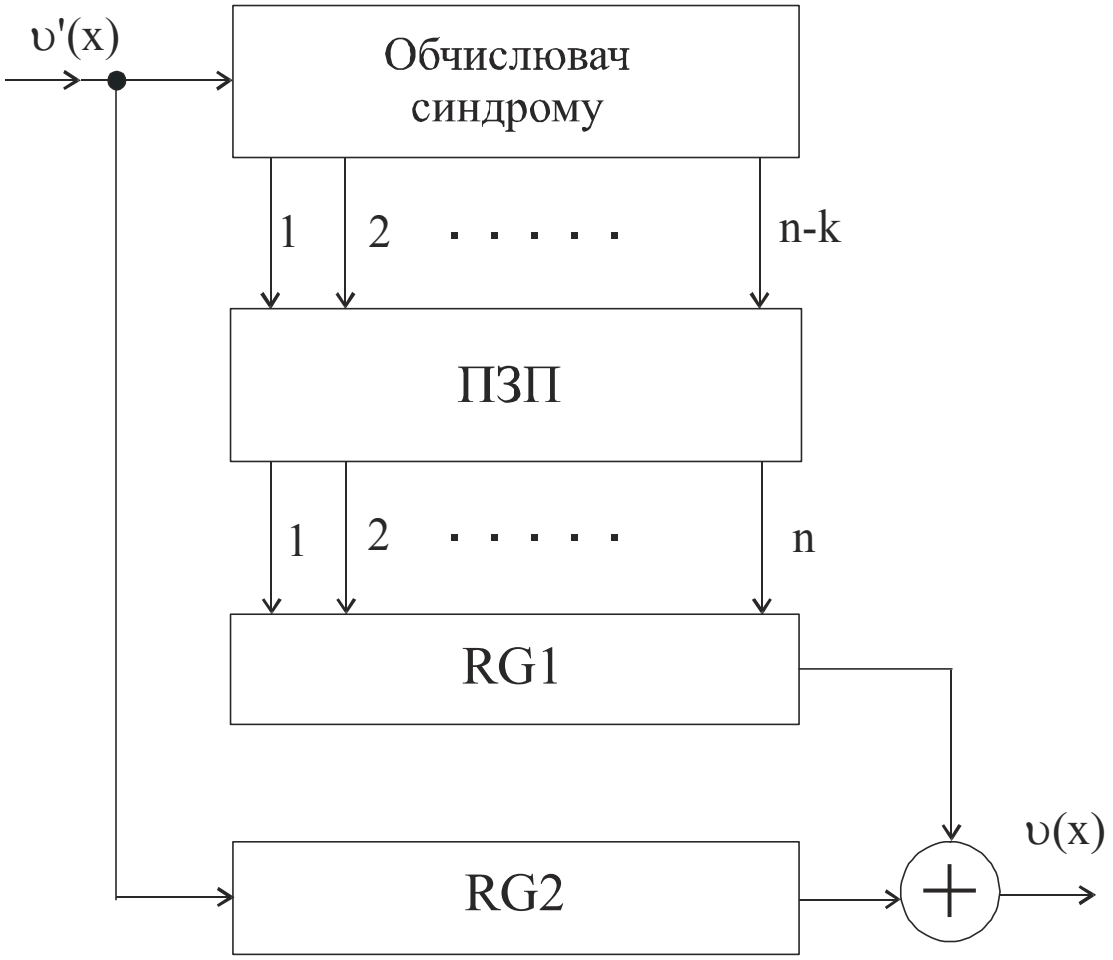


Рисунок 3.7 – Структурна схема декодування коду Ріда-Соломона

У середовищі проектування Quartus II для кожного блоку функціональної схеми наводиться ряд наступних елементів: LogicCells (загальна кількість осередків для елемента); LC Resisters (загальна кількість регістрів в одній комірці); LUT (OnlyLCs - кількість елементів комбінаторної логіки, що входять в комірку); Register - OnlyLCs (кількість елементів реєстрової логіки); LUT / Register LCs (кількість елементів комбінаторної або реєстрової логіки) [30].

Кількість логічних комірок необхідних для блоку, який реалізує код Хемінга на FPGA фірми Altera Cyclone III, що становить менше 1% кристала. Total logic elements 147/39600 (<1%) (таблиця 3.2).

Результати, отримані на етапі синтезу при використанні бібліотеки Altera для блоку, який реалізує код Ріда - Соломона.

Кількість займаних логічних комірок для блоків, які реалізують код CRC і код Хемінга на FPGA фірми Altera Cyclone III, становить менше 1% кристала, що показує їх простоту реалізації і малу кількість витрат ресурсів при розробці (таблиця 3.3). Код Ріда - Соломона займає 2692 елементів з 39600, що становить 7% кристала. Такі апаратні затрати коду Ріда - Соломона пояснюються його ефективністю роботи (таблиця 3.4).

В результаті синтезу була отримана оцінка логічних елементів, яку займає кожний блок в ПЛІС.

Для подальшої розробки їх «в кремнії» потрібно провести синтез в бібліотеці TSMC по нормам 90 нм, для отримання оцінки затримок, а також площі блоків на кристалі.

Очевидно, що блоки, які реалізують кожний з розглянутих алгоритмів, повинні працювати «в зв'язці» з пристроєм обробки інформації. Таким пристроєм було вибрано мікропроцесорне ядро KM211.

При синтезі було встановлено обмеження на часові затримки 1 нс. В результаті синтезу при заданих умовах для всіх блоків затримка склала не більше 1 нс. Це означає, що можливе використання блоку, що реалізує кожен з розглянутих кодів, як окремого функціонального блоку в мікроконтролері.

Таблиця 3.2 – Результат синтезу для блоку, який обчислює код CRC для одного байту даних, слова та подвійного слова

Hierarchy Node	Logic Cells	Logic Registers	LUT-Only LCs	Registers-Only LCs	LUT/Registers LCs
CRC	242	36	206	4	32

Таблиця 3.3 – Результат синтезу для коду Хемінга

Hierarchy Node	Logic Cells	Logic Registers	LUT-Only LCs	LUT/Registers LCs
Hamming	147	2	145	2
-decod	41	0	41	0
-encod	27	0	27	0

Таблиця 3.4 –Результат синтезу для коду Ріда-Соломона

Hierarchy Node	Logic Cells	Logic Registers	LUT-Only LCs	Memory Bits	Registers-Only LCs	LUT/Registers LCs
RS_top	2692	1409	1168	54	211	1198
-decod	2342	1201	1141	54	202	999
-encod	350	208	27	0	9	199

Площа блоку Ріда-Соломона становить 19000 мкм^2, що майже в 15 разів більша за площу, яку займає блоком, які реалізують код Хемінга. Це займає досить велику площу кристала, що впливає на суму при замовленні мікроконтролера. Для прикладу, площа ядра KM211 сімейства, яку займає в кремнії, становить 90000 мкм^2.

У роботі були досліджені методи кодування для підвищення надійності зберігання інформації, а саме: циклічний надлишковий код (CRC код), код Ріда-Соломона і код Хемінга.

CRC код є вдалим рішенням в тих випадках, коли корекція помилок не потрібно і достатньо лише перевірити, чи успішно записані дані [3]. Цей метод володіє такими перевагами, як невисокі витрати ресурсів, простота реалізації в апаратних пристроях, а також готовий сформований математичний апарат з теорії лінійних циклічних кодів. Тому цей код є популярним, а також хорошим засобом для виявлення помилок на практиці, наприклад, коли виникають помилки через наявність в каналі передачі даних шуму.

Коди Хемінга - це найбільш відомі з самоконтролюючих і самокорегуючих кодів [3].

Код Хемінга здатний виявити і виправити одноразову помилку. Існує модифікований код Хемінга, здатний виправляти одиночну і виявляти подвійну помилки.

Коди CRC і Хемінга на відміну від Ріда-Соломона використовують менше апаратних затрат, а також площі в кристалі, що дозволяє в подальшому реалізувати їх у вигляді окремого функціонального блоку для мікроконтролера. Такі коди можна використовувати для підвищення надійності зберігання ключової інформації, наприклад, в банківських картах. При цьому дане поліпшення не буде сильно впливати на складність пристрою і зручність користування карткою.

Код Ріда-Соломона є найбільш потужним кодом, який виправляє багаторазові пакети помилок.

На даний момент коди Ріда - Соломона мають дуже широку сферу застосування завдяки їх здатності знаходити і виправляти багаторазові пакети помилок. Їх можна застосовувати для виправлення помилок в багатьох системах: пристроях пам’яті, бездротових або мобільних комунікаціях, супутникових комунікаціях, цифровому телебаченні. Проте, процеси кодування і декодування вимагають великих затрат, що сильно обмежує область його застосування.

ВИСНОВКИ

В магістерській роботі розв’язано актуальну задачу розробки та дослідження алгоритмів надійного розподіленого зберігання даних на основі корегуючих кодів. При цьому отримано наступні результати.

1. Проведено аналіз методів побудови систем надійного зберігання даних. проаналізовано типи завад в каналах зв’язку та їх вплив на достовірність передавання даних.

2. Розкрито теоретичні основи завадостійкого кодування, проведено аналіз найбільш поширених завадостійких кодів, зокрема: циклічні коди, коди в системі залишкових класів, коди БЧХ, коди Ріда-Соломона. Приведено основні параметри та характеристики вказаних кодів.

3. Проведено дослідження методів кодування та декодування, а також розроблено структурні та функціональні схеми пристроїв кодування та декодування даних.

4. Найбільш ефективними, для відновлення даних, є коди Ріда – Соломона. Час декодування коду Ріда – Соломона є значно меншим, ніж у інших кодів з такою ж довжиною блоку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Abughalieh, N., Steenhaut, K., Nowé, A. Low power channel coding for wireless sensor networks. Communications and Vehicular Technology in the Benelux (SCVT), 2010 17th IEEE Symposium on. IEEE, 2010. – P. 1-5.

2. Agarwal R., Popovici E. M., O'Flynn B. Adaptive wireless sensor networks: A system design perspective to adaptive reliability // In Wireless Communications and Sensor Networks. – 2006. – P. 216-225.

3. Akyildiz F., Pompili D., Melodia T. Underwater Acoustic Sensor Networks: Research Challenges," Ad Hoc Net., vol. 3, no. 3, May 2005. – P. 257–79.

4. Alrajeh N. et al. Error Correcting Codes in Wireless Sensor Networks: An Energy Perspective. Applied Mathematics & Information Sciences, 2015, 9.2. – P.809-818.

5. Al-suhail G. A., Louis K. W., Abdallah T. Y. Energy Efficiency Analysis of Adaptive Error Correction in Wireless Sensor Networks. International Journal of Computer Science Issues, Vol. 9, Issue 4, No 2, July 2012 – P. 79–84.

6. Al-suhail G. A., Louis K. W., Abdallah T. Y. Energy Efficiency Analysis of Adaptive Error Correction in Wireless Sensor Networks. International Journal of Computer Science Issues, Vol. 9, Issue 4, No 2, July 2012 – P. 79–84.

7. Ang Li-minn, et al. Wireless Multimedia Sensor Networks on Reconfigurable Hardware. Springer, 2013 – 283 p.

8. Antonio de la Piedra, An Braeken, Abdellah Touhafi. Sensor Systems Based on FPGAs and Their Applications: A Survey. Sensors, 2012, Vol.12 (9). – P.1235 – 1264.

9. Balakrishnan Gopinath, et al. Performance analysis of error control codes for wireless sensor networks. Information Technology, 2007. ITNG'07. Fourth International Conference on. IEEE, 2007. – P. 876–879.

10. Bao X., Li J. Matching Code-on-Graph with Network-on-Graph: Adaptive Network Coding for Wireless Relay Networks // IEEE Transactions on Wireless Communications, 2008. – V. 7, N. 2. – P. 574–583.

11. Bin Qaisar S., Karande S., Misra, K., Radha, H. Optimally mapping an iterative channel decoding algorithm to a wireless sensor network. In Communications, 2007. ICC'07. IEEE International Conference on IEEE 2007. – P. 283-288.

12. Chlamtac I., Petrioli C., Redi J. Energy-conserving go-back-N ARQ protocols for wireless data networks. In Proceedings of IEEE ICUPC'98, Piscataway, NJ, USA, October 1998ю– Volume 2. – P. 1259–1263.

13. Chlamtac I., Petrioli C., Redi J. Energy-conserving selective repeat ARQ protocols for wireless data networks // Personal, Indoor and Mobile Radio Communications, 1998. The Ninth IEEE International Symposium on. – IEEE, 1998. – T. 2. – P. 836-840.

14. Chouhan, Sonali, Ranjan Bose, and M. Balakrishnan. Integrated energy analysis of error correcting codes and modulation for energy efficient wireless sensor nodes. Wireless Communications, IEEE Transactions on 8.10. – 2009. – P.348-355.

15. Croce S., Marcelloni F., Vecchio M. Reducing power consumption in wireless sensor networks using a novel approach to data aggregation. The Computer J., 2008. – Vol. 51, No. 2. – P. 227 – 239.

16. Fasolo E., Rossi M., Widmer J., Zorzi M. In-network aggregation techniques for wireless sensor networks: a survey. IEEE Trans. Wireless Commun, 2007. – Vol. 14, No. 2. – P. 70–87.

17. Goh Vik Tor, and Mohammad Umar Siddiqi. "Multiple error detection and correction based on redundant residue number systems."Communications, IEEE Transactions on 56.3, 2008. – P.325-330.

18. Gray A., Lee C., Arabshahi P., Srinivasan J. Object-oriented reconfigurable processing for wireless networks. In: IEEE International Conference on Communications (ICC 2002), 2002. – Vol. 1. – P. 497–501.

19. Guestrin C, Bodik P, Thibaux R, Paskin M, Madden S. Distributed regression: an efficient framework for modeling sensor network data. In: International conference on information processing in sensor networks (IPSN). Third International Symposium IEEE, 2004. – P. 1-10.

20. Howard, S. L., Schlegel, C., & Iniewski, K. Error control coding in low-power wireless sensor networks: When is ECC energy-efficient?. EURASIP Journal on Wireless Communications and Networking, 2006.– № 2.– P.29-29.

21. Jeong J., Cheng Tien Ee. Forward error correction in sensor networks. University of California at Berkeley, 2003. – P. 1-13.

22. Linjun Mei, Dan Feng, Jianxi Chen, Lingfang Zeng, Jingning Liu. A Write-Through Cache Method to Improve Small Write Performance of SSD-Based RAID. Networking Architecture and Storage (NAS). International Conference on, 2017. – P.1-6.

23. Machida, F., Xia, R., Trivedi, K. S. Performability Modeling for RAID Storage Systems by Markov Regenerative Process. IEEE Transactions on Dependable and Secure Computing, 2018, 15(1), Pp. 138-150.

24. Nykolaychuk Ya. M. Theoretical Foundations for the Analytical Computation of Coefficients of Basic Numbers of Krestenson's Transformation / Ya. M. Nykolaychuk, M. M. Kasianchuk, I. Z. Yakymenko // Cybernetics and Systems Analysis. – 2014.– Vol. 50. – P. 649-654.

25. Roshanzadeh M., Saqaeeyan S. Error Detection & Correction in Wireless Sensor Networks By Using Residue Number Systems. International Journal of Computer Network and Information Security (IJCNIS) 4.2, 2012. – №2. – P. 29-35.

26. Rui Ye, Wentao Meng, Shenggang Wan. Extending Lifetime of SSD in Raid5 Systems through a Reliable Hierarchical Cache. International Conference on Networking Architecture and Storage (NAS). – P.1-8, 2017.

27. Srivastava S., Spagnol C., Popovici E. Analysis of a set of error correcting schemes in multi-hop wireless sensor networks. In Research in Microelectronics and Electronics, PRIME 2009, 2009. – P. 1-4.

28. Zoubek, C., Seufert, S., & Dewald, A. Generic RAID reassembly using block-level entropy. Digital Investigation, 2016, 16, P.44-54.

29. Блейхут Р. Теория и практика кодов контролирующих ошибки. Пер. с англ. – М.: Мир, 1986. – 576 с.

30. Дунець Р.Б., Тиранський Д.Я. Проблеми побудови частково реконфігурованих систем на ПЛІС // Радіоелектронні і комп'ютерні системи. – 2010. – №7(48). – С.200 – 204.

31. Зюко А.Г. Помехоустойчивость и эффективность систем передачи информации. – М.: Радио и связь, 1985. – 272 с.

32. Зяблов В.В., Шавгулидзе С.А. Обобщенные каскадные помехоустойчивые конструкции на базе сверточных кодов. – М.: Москва, 1991. – 207 с.

33. Мак-Вильямс Ф. Дж., Слоэн Н. Дж. А. Теория кодов исправляющих ошибки. Пер. с англ. – М.: Связь, 1979. –744 с.

34. Максфилд, К. Проектирование на ПЛИС. Курс молодого бойца // М.: Издательский дом «Додэка-XXI», 2007. — 408 с.

35. Матіїшин Ю.С., Харчун В.В., Гевко Я.І. Система надійного розподіленого зберігання даних на основі коригуючих кодів. Матеріали семінару комп'ютерні науки та інформаційні технології, CSIT'2018, Тернопіль, 2 червня 2018 р., С.48.

36. Методичні рекомендації до виконання магістерської роботи з освітнього ступеня "Магістр". Спеціальність: 123 - Комп'ютерна інженерія. Магістерська програма - Комп'ютерна інженерія" / О.М. Березький, Л.О. Дубчак, Г.М. Мельник /Під ред. О.М. Березького – Тернопіль: ТНЕУ, 2018.– 41 с.

37. Муттер В.М. Основы помехоустойчивой телепередачи информации. – Л.: Энергоатомиздат. Ленингр. отд-ние, 1990. – 268 с.

38. Нейфах А.Э. Сверточные коды для передачи дискретной информации. – М.: Наука, 1979. – 222 с.

39. Поляков, А. К. Языки VHDL и Verilog в проектировании цифровой аппаратуры на ПЛИС // М.: МЭИ, 2012. — 221 с.

40. Скляр Б. Цифровая связь. Теоретические основы и практическое применение, 2-е издание.: Пер. с англ. – М. : Издательський дом “Вильямс», 2003. – 1104 с.

41. Столлингс В. Беспроводные линии связи и сети.: Пер. с англ. – М.: Издательський дом «Вильямс», 2003. – 640 с.

42. Шувалов В.П., Захарченко Н.В., Шварцман В.О. и др. Передача дискретных сообщений: Учебник для вузов / Под ред. В.П . Шувалова. – М.: Радио и связь, – 1990 – 464 с.

43. Яцків В.В. Виявлення та виправлення багатократних помилок на основі модулярних коректуючих кодів / В.В. Яцків // Інформаційні технології та комп’ютерна інженерія. – 2015. – Том 33, №2. – С.77-82.

44. Яцків В.В. Виявлення та виправлення багатократних помилок на основі модулярних коректуючих кодів / В.В. Яцків // Інформаційні технології та комп’ютерна інженерія. – 2015. – Том 33, №2. – С.77-82.

45. Яцків В.В. Двовимірні коректуючі коди на основі модулярної арифметики / В. В. Яцків, Т.Г. Цаволик // Вісник Хмельницького національного університету. Технічні науки. – 2015. – № 4 (227). – С.144 - 148.

46. Яцків В.В. Завадозахищений метод передавання даних в безпровідних сенсорних мережах / В.В.Яцків, Н.Г.Яцків, Д.І.Боднар // Науковий вісник Чернівецького університету: Комп’ютерні системи та компоненти. – Чернівці: ЧНУ. – 2009. – Вип. 446. – С.117-120.

47. Яцків В.В. Контроль виконання арифметичних операцій на основі модулярних кодів /В.В.Яцків // Вимірювальна та обчислювальна техніка в технологічних процесах. – 2015. – № 4(53). – С.135-138.

48. Яцків В.В. Метод завадостійкого кодування даних на основі модулярних коректуючих кодів / В.В.Яцків, Т. Г.Цаволик // Матеріали V –ї міжнародної науково-практичної конференції молодих вчених

«Інфокомунікації – сучасність та майбутнє» (29-30 жовтня 2015 р., м. Одеса). – Одеса, ОНАЗ, 2015. – С. 111-114.

49. Яцків В.В. Метод завадостійкого кодування даних на основі модулярних коректуючих кодів / В.В.Яцків, Т. Г.Цаволик // Матеріали V –ї міжнародної науково-практичної конференції молодих вчених «Інфокомунікації – сучасність та майбутнє» (29-30 жовтня 2015 р., м. Одеса). – Одеса, ОНАЗ, 2015. – С. 111-114.

50. Яцків В.В. Модифіковані коректуючі коди системи залишкових класів та їх застосування / В.В. Яцків // Інформаційні технології та комп'ютерна інженерія. – 2013 – №2. – С.39-45.

51. Яцків В.В. Принципи реалізації основних модулярних операцій на ПЛІС / В.В. Яцків // Матеріали VI – ої Українсько – польської науково-практичної конференції «Електроніка та інформаційні технології» ЕлІТ-2014, Львів – Чинадієво, Україна. – С.20-22.