# **Визначення відстані Хеммінга в розподілених інфраструктурах для пошуку подібності.**

**Аннотація**. Сьогодні обсяги даних доступних в Інтернеті перевищують зетабайти (ZB). Ця умова визначає явище яке відоме в літературі як Big Data. Хоча традиційні бази даних дуже ефективні для пошуку та отримання потрібного контенту, вони не є ефективними в умовах роботи з великими обсягами даних, оскільки велика частина цих даних є неструктурованою та розподіленою в Інтернет. Таким чином для забезпечення пошуку подібності необхідні нові бази даних. Для того аби справлятися з даною ситуацією в даному розділі пропонується досліджувати схожість за Хеммінгом між ідентифікаторами контенту, що згенеровані з використанням функції випадкової гіперплощини хешування. Такі ідентифікатори забезпечують основу для побудови розподілених інфраструктур, що спрощує пошук подібності. В цьому розділі м представимо два різні підходи: P2P (Peer-to-Peer) (Hamming DHT (Distributed Hash Table)) та Data Center (HCube). Представлені оцінки що вказують на те, що обидва підходи здатні покращити результати пошуку подібності.

## **1. Вступ**

Згідно сучасного розуміння явища Big Data, користувачі стали джерелами даних, компанії зберігають незліченні обсяги інформації про клієнтів, мільйони сенсорів отримують інформацію з середовища, створюючи та обмінюючись даними в Інтернеті речей. Згідно дослідження International Data Corporation (IDC), опублікованого в травні 2010 [1], кількість даних доступних в Інтернеті перевищує 2 ZB в 2010, подвоююись кожні 2 роки і може перевищити 8 ZB в 2015. Дослідження також показало що близько 90% цих даних складаються з неструктурованих, гетерогенних та різних за природою даних як тексти, зображення та відео.

Нові технології такі як Hadoop [2] та MapReduce [3] є прикладами рішень створених щоб вирішувати проблеми які накладає Big Data – обсяг, різноманітність та швидкість. Використовуючи пераленьні обчислення в поєднанні з грід технологіями, чи, як нещодавно, використовуючи інфраструктуру хмарних обчислень, IT організації пропонують засоби для роботи з широкомасштабними, розподіленими та інформаційномісткими (такими що використовують великі об’єми даних) задачами. Зазвичай такі технології пропонують розподілену файлову систему та автоматизовані інструменти для налаштування в процесі роботи (“на льоту”) кількості серверів залучених в обробку задач. В цьому випадку великі об’єми даних пропускаються через мережу і підключаючись до серверів передають пари <ключ, значення> з маперів до ред'юсорів щоб отримати необхідні результати. В цьому випадку бажано мінімізувати потреби в передачі даних по мережі щоб пришвидшити роботу всієї обчислювальної задачі.

В той час як дані рішення є, беззаперечно, ефективними для роботи традиційних застосувань, як, наприклад, пакетної обробки великих об’ємів даних, вони не забезпечують достатню підтримку пошуку подібності [4], ціллю якої є вилучення наборів подібних даних за даним рівнем подібності. Наприклад, схожість даних може бути використана в системі рекомендацій заснованій на інформації соціальних профілів користувачів. В даному прикладі профіль користувача може бути визначених як набір характеристик що унікально впливають на те як користувачі приймають рішення. Користувачі зі схожими характеристиками, скоріш за все, матимуть схоже інтереси та вподобання.

Таким чином, щоб зробити систему пошуку подібності засновану на даних профілів користувачів, характеристики користувачів в соціальній мережі можуть бути поміщені в вектор і, використовуючи Vector Space Model (VSM), подібність між користувачами може бути визначена використовуючи метрики відстаней між векторами як, наприклад, Евклідова відстань, косинус між векторами та Хеммінгова відстань. Але, за виключенням Хеммінгової відстані, інші метрики піддаються прокляттю розмірності (значний ріст необхідних обчислень при рості розмірності задачі) [4]. Великі обчислювальні витрати внаслідок розмірності задачі – це виклик що постає перед системами поуку подібності в випадку Big Data.

В цьому розділі описано як підтримувати пошук подібності використовуючи Хеммінгову відстань як метрику подібності. Щоб це досягти, дані індексуються в базі даних використовуючи LSH (Locality Sensitive Hashing) функцію що називається функцією RHH (Random Hyperplane Hashing) [5]. RHH – це сімейство LSH функцій, що використовують косинус між векторами та Хеммінгову відстань як метрику відстані між згенерованими двійковими рядками, тобто чим більший косинус між парою векторів контенту, тим менша Хеммінгова відстань між двійковими рядками. Ці двійкові рядки представляють ідентифікатори даних чия подібність можу бути виміряна Хеммінговою відстанню. Кожен запит в цій базі даних вираховується використовуючи Хеммінгову відстань між ідентифікатором запиту та кожним ідентифікатором даних.

При пошуку подібності запит що обробляється складається з того ж набору характеристик контенту індексованого в базі даних. Кожен користувач системи пошуку подібності вводить бажані характеристики та рівень подібності згідно очікуваного об’єму відповіді на запит. Чим більший рівень подібності тим менша кількість даних отримується внаслідок більшої специфічності (націленості) запиту. Запит індексується в базі даних з використанням RHH функції та подібності за Хеммінгом між запитом та всіма обчисленими ідентифікаторами даних в базі даних. Всі профілі чия подібність за Хеммінгом задовольняє бажаному рівню подібності повертаються у відповідь на запит.

Щоб оцінити систему пошуку подібності були проведені наступні тести: взаємозв’язок подібності з використанням косинусу між векторами контенту та подібності за Хеммінгом їх ідентифікаторів представлені використовуючи чотири рівні подібності (0.7, 0.8, 0.9 та 0.95); частотних розподіл відстані за Хеммінгом між ідентифікаторами контенту відповідно до їх рівня подібності; також надані результати деяких запитів та відповідей на них. В наших експериментах вектори контенту представляють профілі користувачів в Adult Data Set репозиторію UCI [6].

В нашій попередній роботі [7] було представлено рішення з оверлеями для системи пошуку подібності що було створено на вершині DHT (Distributed Hash Table) структури. Головним чином була показана можливість зберігання подібних даних на серверах близько до логічного простору оверлейної мережі використовуючи примітив put(k,v) і що також можливо ефективно відновлювати набір подібних даних використовуючи один get(k,sim) примітив. В іншій попередній роботі [8] був показаний HCube – рішення для дата центру (центру обробки даних) для підтримки пошуку подібності в випадку Big Data, спрямоване на зменшення відстані щоб відновити схожий контент при пошуку подібності. В HCube подібні дані зберігаються на тому ж сервері чи на серверах розташованих близько в дата центрі.

Даний розділ організований наступним чином: частина 2 показує деякі основи технологій що використовуються в Hamming DHT та HCube. Частина 3 містить огляд літератури на пов’язані роботи з пошуку подібності в P2P (Peer-to-Peer) мережі та дата центрі. Частина 4 коротко показує рішення Hamming DHT та HCube. Частина 5 оцінює запропоновану систему пошуку подібності в розподіленому випадку. В частині 6 надані деякі ремарки та огляд майбутньої роботи.

## **2. Короткі теоретичні відомості**

Ця частина показує концепцію VSM (Vector Space Model) – модель представлення даних в вигляді векторів в багатовимірному просторі, функцію RHH – LSH функцію для генерування ідентифікаторів даних що зберігатимуть подібність між векторами контенту, та функцію подібності за Хеммінгом – функцію подібності що використовується для порівняння відстані за Хеммінгом між двійковими ідентифікаторами.

### 2.1 Модель векторів у просторі (VSM (Vector Space Model))

VSM – алгебраїчна модель представлення об’єктів як векторів. В загальному випадку, кожен вимір (кожна координата) цих векторів пов’язаний з характеристикою самого контенту як, наприклад, ключові слова в тексті, гістограма кольорів на зображенні чи атрибути профілю в соціальній мережі.

Набір векторів отриманих з Adult Data Set репозиторію UCI [6], використовується щоб описати процедуру перетворення таких атрибутів в вектор що може бути виміряний та порівняний з іншими векторами використовуючи алгебраїчні операції. В своїй суті цей набір даних містить інформацію про дорослих громадян що живуть в США включаючи наступні атрибути: вік, робочий клас, рівень освіти, кількість років проведених в школі, сімейний стан, вид діяльності, родинні зв’язки, расу, стать, фінансові здобутки та втрати за останній рік, кількість робочих годин на тиждень, країна походження, дохід за рік. Зразки цих профілів:

– ADULT1 - 43; Self-emp-not-inc; 5th-6th; 3;Married-civ-spouse;Craft-repair;Husband; White; Male; 0; 4700; 20; United-States; ≤50K

– ADULT2 - 56; Private; 10th; 6; Married-civ-spouse; Craft-repair; Husband;White; Male; 0; 0; 0.45; France; ≤50K

– ADULT3 - 50; Self-emp-inc; Prof-school; 15; Married-civ-spouse;Prof-specialty; Husband;White; Male; 0; 0; 36; United-States; ≥50K

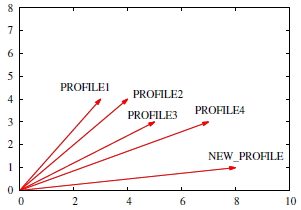
– ADULT4 - 30; Private; Prof-school; 15; Married-civ-spouse; Prof-specialty; Husband; White; Male; 0; 0; 30; United-States; ≥50K

Для експериментів повинні бути зроблені деякі перетворення над цими векторами. Числові атрибути представлені в векторах, такі як “вік”, повинні бути нормалізовані до границь [0..1]. Така нормалізація була зроблена шляхом ділення значення на найбільше значення в наборі. Координати що представляють дискретні атрибути (наприклад, стать, що може бути “чоловіча” чи “жіноча”) були розділені в 2 різні координати, кожна з яких в окремому вимірі відповідно до можливих значень. Для атрибуту “стать” було створено два виміри “чоловіча” та “жіноча”. Якщо людини – чоловік, відповідний вектор має значення “1” для виміру “чоловіча” та “0” для виміру “жіноча” і навпаки, якщо людина – жінка.

Як показано в [9] ці процедури були необхідні тому що поняття подібності чи відстані для дискретних даних не таке очевидні як для числових і тут це було основною задачею.Це виникає внаслідок того що різні значення дискретного атрибуту не упорядковані по свій суті і, таким чином, поняття впорядкованості для них не визначене. Також поняття подібності може відрізнятися в залежності від конкретного домену. Внаслідок цього кожен атрибут вектору має бути розширений на кількість вимірів відповідно до всіх значень які він [атрибут] містить. Використовуючи цю процедуру, вектор повинен бути розширений з 14 до 103 вимірів.

Можна виміряти подібність між векторами що представляють дані шляхом вирахування косинусу кута між ними (simcos). Як показано в [10], подібність за косинусом дає високоякісні результати в декількох доменах. Щоб показати це, на мал. 1 відображено застосування в якому профілі користувачів представлені двовимірними векторами, кожен вимір описує інтерес користувача до спорту та літератури, використовуючи масштаб в якому “0” означає відсутність інтересу, а “10” – повний інтерес до області. Розглянемо чотири профілі користувачів представлені кортежами PROFILE1(3,4) – ранг 3 для спорту та 4 для літератури, PROFILE2(4,4) – ранг 4 для спорту і літератури, PROFILE3(5,3) – ранг 5 для спорту та 3 для літератури та PROFILE4(7, 3) – ранг 7 для спорту та 3 для літератури.

Щоб дати розуміння використання подібності за косинусом (simcos), розглянемо розробку системи рекомендації друзів засновану на профілях користувачів відповідальну за вказування на друзів з подібними інтересами до профілю нового користувача представленого в вигляді кортежу NEW PROFILE(8,1) – ранг 8 для спорту та 1 для літератури. В цьому випадку система рекомендації запропонує користувачу NEW PROFILE наступний порядок для встановлення нових знайомств: 1) PROFILE4 чий simcos ≈ 0.96, 2) PROFILE2 чий simcos ≈ 0.9, 3) PROFILE3 чий simcos ≈ 0.8, and 4) PROFILE1 чий simcos ≈ 0.7.



Спорт

Мал. 1 Графічне представлення векторів профілей

### 2.2 RHH (Random Hyperplane Hashing) та подібність за Хеммінгом

Функції LSH зменшують розмірність векторів що представляють дані забезпечуючи те, що чим більш подібними два об’єкти є, тим більш подібними будуть хеш-значення їх векторів [4]. Кожне сімейство LSH функцій відноситься до певної функції подібності. RHH – приклад сімейства LSH функцій пов’язаних з подібністю за косинусом. В даному контексті, Чарікар [5] представляє техніку хешування резюмовану в даній частині.

Дано набір  з m векторів  кожна з координат яких випадковим чином отримана з стандартного нормального розподілу, та вектор . Хеш-функція  визначається наступним чином:



Для кожного  генерується один біт, а результат m об’єднується щоб створити m-бітний хеш-ключ для даного вектора . Для двох векторів даних  ймовірність згенерувати подібні ключі – значення косинусу кута між  та . Відповідно, чим більша подібність за косинусом, тим більш ймовірно що згенеровані ключі міститимуть спільні біти, приводячи схожість двох ідентифікаторів в відстані за Хеммінгом (тобто в кількості різних біт в двох двійкових рядках). І навпаки – подібність за Хеммінгом можна вирахувати як , де simh – подібність за Хеммінгом, Dh – Хеммінгова відстань, m – число біт в рядку що представляє ідентифікатор профілю користувача.

Наприклад, нехай застосування використовує 8-бітні ідентифікатори, повинна бути згенерована послідовність m=8 випадкових векторів  і повернені m  об'єднаних бітів щоб згенерувати 8-бітний ідентифікатор. В таблиці 1 показано приклад з порівнянням подібності за косинусом (simcos), відстані за Хеммінгом (Dh) та подібністю за Хеммінгом (simh) між 4 типовими 8-бітними ідентифікаторами профілів користувачів. В цьому прикладі “PROFILE B”, що найбільш подібний до “PROFILE A” на основі подібності за косинусом також має найменшу відстань за Хеммінгом (Dh) відповідну подібності за Хеммінгом (simh) в 0.875.

Таблиця 1. 8-бітні ідентифікатори профілів, відстань за Хеммінгом (Dh), подібність за Хеммінгом (simh) та подібність за косинусом (simcos) профілів користувачів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Профілі користувачів | 8-бітні ідентифікатори | Dh | simh | simcos |
| PROFILE B,  PROFILE A | 01001010,  01101010 | 1 | 0.875 | 0.99 |
| PROFILE C,  PROFILE A | 01001000,  01101010 | 2 | 0.75 | 0.95 |
| PROFILE D,  PROFILE A | 01011000,  01101010 | 3 | 0.625 | 0.85 |

## **3. Огляд літератури**

Серед пов’язаних праць на тему пошуку подібності доступних в літературі, більшість засновані на певному виді схем індексації, таких як, наприклад, функції хешування [4] чи SFC (Space Filling Curve) [11]. Обидві мотивовані проблемою найближчих сусідів [4], тобто як вилучити найбільш подібні дані в індексованому просторі. Використання властивості подібності за Хеммінгом RHH функції надає перевагу: немає необхідності використовувати криву Гільберта для об’єднання подібних ідентифікаторів.

Як показано в [4], SFC піддаються прокляттю розмірності [4]. Для вирішення проблеми розмірності, Індик [4] запропонував використовувати LSH функції. Ці функції можуть зменшити кількість вимірів вектору, створюючи ідентифікатор для нього, представлений двійковим рядком розміру m (m>=0). Відстань між двома двійковими рядками згенерованими застосуванням LSH функції пари векторів контенту обернено пропорційна їх схожості. Наша пропозиція використовувати відстань за Хемміном для виміру подібності контенту, що відображається на відповідну схожість профілів користувачів, на скільки нам відомо, -- новий підхід в створенні систем пошуку подібності.

В першій реалізації прототипу пошуку подібності було необхідно проводити пошук по всій базі даних. Ми знаємо, що цього недостатньо в випадку Big Data, що мотивувало дану роботу. Для того аби розв’язати це завдання в випадку Big Data, ми пропонуємо дві розподілені архітектури: Hamming DHT [7] та HCube [8], що можуть виступати як інфраструктури для пошуку подібності в контексті Big Data. Hamming DHT – це P2P мережа що використовує подібність за Хеммінгом щоб полегшити пошук та вилучення подібних профілів. HCube – інфраструктура дата центру що спеціалізується на пошуку подібності використовуючи подібність за Хеммінгом.

Оверлейні підходи з’явилися як рішення для допомоги керування великими об’ємами даних. В основному, ці рішення засновані на DHT (Distributed Hash Tables), поширенні даних між вузлами оверлейної мережі. Згідно [12], з’явилися багатовимірні методи індексації P2P як нова парадигма за останні декілька десятиліть. В цьому випадку DHT повинні бути обладнані багатовимірними запитами та можливостями для обробки пошуку подібності.

Hycube [13] – приклад DHT що використовує відстань за Хеммінгом як метрику відстані та організовує вузли в гіперкуб одиничного розміру. Однак витрати пов’язані з підтримкою гіперкуба та неповних кубів більші ніж витрати на послідовне хешування DHT, як в запропонованій Hamming DHT.

pSearch, запропонований Тангом та ін. [14] – P2P мережа що також використовує вектори контенту та подібність за косинусом, але відрізняється від Hamming DHT, оскільки спеціалізується на пошуку подібності текстових документів в P2P мережі. Також в pSearch обговорюється шлях побудови розподіленого словника термінів для індексації документів.

Бхаттачарія [15] використовує подібність за косинусом та LSH функції щоб запропонувати фреймворк для пошуку подібності в розподілених базах даних. Це розширює get(k) примітив, наданий будь-якою реалізацією DHT для підтримки рівня подібності get(k,Dh) на відстані за Хеммінгом Dh від k. Ця пропозиція не націлена на пошук подібності, оскільки це надбудова над будь-якою існуючою DHT для підтримки пошуку подібності.

З цього короткого огляду ми можемо виділити що Hamming DHT зосереджена на зменшенні відстані (в стрибках) необхідної для вилучення подібного контенту і на підвищенні ефективності пошуку в системах пошуку подібності . Використання LSH функцій для індексації подібного контенту – не нове, але дослідження подібності за Хеммінгом в організації ідентифікаторів контенту – оригінальна ідея. Ця модифікація робить Hamming DHT простішим за інші рішення.

Порівнюючи ці дві роботи, HCube – розподілене рішення що використовує такі методи як LSH функції та SFC для підтримки пошуку подібності, але HCube не засноване на оверлейному рішенні. HCube представляє серверно-орієнтовану структуру дата центрів що спеціалізується на пошуку подібності, де подібні дані зберігаються на фізично близьких серверах, з вилученням цих даних за зменшену кількість стрибків і з меншими вимогами до обробки.

З іншого боку, такі гравці як Google [16] та Amazon [17] створили дата центри що спеціалізуються на зберіганні та обробці великих об’ємів даних використовуючи рішення MapReduce, але вони не розглядаються для пошуку подібності. Таким чино HCube відкриває нове поде для досліджень де застосування можуть скористатися своєю структурою для пошуку подібності, уникаючи задач інтенсивної обробки як з’являються в MapReduce, оскільки подібні дані організовані на серверах що близько розташовані протягом фази сортування.

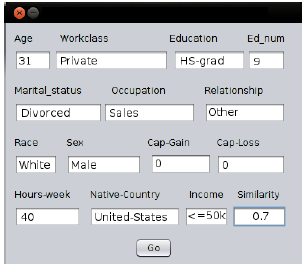
В наступній частині показано як здійснювати пошук в розподілених інфраструктурах заснованих на відстані за Хеммінгом, розглядаючи схожість як результат.

## **4. Пошук подібності заснований на відстані за Хеммінгом**

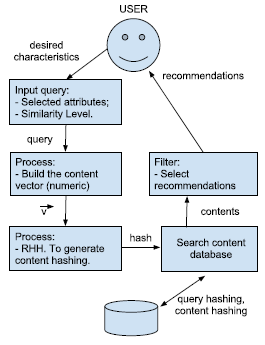
В запропонованій системі пошуку подібності, користувач що створює запит повинен обрати бажані характеристики та рівень подібності в інтервалі [0..1] для запиту. Потім відбувається пошук в контенті бази даних на предмет записів що задовольняють обраним характеристикам. В першій реалізації дозволені тільки повні запити з заповненими всіма атрибутами.

На мал. 2 показано приклад головного інтерфейсу що використовується для вибору бажаних характеристик для пошуку. В цьому ж інтерфейсі користувач вводить рівень подібності. Як було вказано раніше, чим більший рівень подібності, тим менша кількість отриманих результатів, тому що менше профілів задовольняють запиту. Результати організовані в список як результат пошуку подібності.

Мал. 3 резюмує операції прототипу. Користувач запропонованої системи пошуку подібності обирає всі бажані характеристики для запиту і бажаний рівень подібності. Після вибору бажаних характеристик, створюється вектор контенту за процедурою описаною в 2.1. Цей вектор  використовується як вхідні дані для RHH функції що генерує ідентифікатор хешу запиту. Хеммінгова відстань між  та всіма записами бази даних вираховується і всі записи які задовольняють обраному рівню подібності виводяться користувачеві. Може бути застосований фільтр другого рівня для, наприклад, покращення результатів чи вибору найновіших з них, але цей фільтр не представлений в даній реалізації.



Мал. 2. Прототип інтерфейсу для запиту



Мал. 3. Операції прототипу

Деякі приклади результатів, отримані використовуючи набір даних описаний в частині 2.1, представлені та проаналізовані. Прототип інтерфейсу використовувався для отримання результатів всіх подібних профілів в базі даних згідно обраного рівню подібності. Для представлення результатів було використано чотири рівня подібності 0.7, 0.8, 0.9 та 0.95.

В таблиці 2 представлені деякі результати. В усіх випадках вхідними даними для пошуку був профіль з наступними характеристиками: “31” рік; працює в “Private” секторі; закінчив школу, “HS-grad”; “9” років провів у школі; “Divorced”; працює в відділі “Sales”; родинні зв’язки – “Other-relative”; “White”; “Male”; народився в “United-States”; дохід на рік “<=50K” американських доларів. Був використаний рівень подібності 0.7, що означає що були отримані результати з подібність 0.7 і більше.

Таблиця 2. Деякі результати прототипу та їх подібність за Хеммінгом (simh) з використанням профілю користувача q як вхідні дані. q: <31; Private; HS-grad; 9; Divorced; Sales; Other-relative; White; Male;

United-States; <=50K>.

|  |  |
| --- | --- |
| R1, simh = 0.75 | <41; Private; HS-grad; 9; Married-civ-spouse; Adm-clerical; Wife; White; Female; United-States; >50K> |
| R2, simh = 0.7266 | <46; Private; Some-college; 10; Married-civ-spouse; Sales; Husband; White; Male; United-States; >50K> |
| R3, simh = 0.7734 | <36; Private; Some-college; 10; Married-civ-spouse; Craft-repair; Husband; White; Male; United-States; <= 50K> |
| R4, simh = 0.8359 | <22; Private; HS-grad; 9; Never-married; Other-service; Other-relative; White; Male; United-States; <=50K> |
| R5, simh = 0.8516 | <53; Private; HS-grad; 9; Married-civ-spouse; Craft-repair; Husband; White; Male; United-States; <=50K> |
| R6, simh = 0.8438 | <37; Private; HS-grad; 9; Separated; Handlers-cleaners; Not-in-family; White; Male; United-States; <=50K> |
| R7, simh = 0.9062 | ¡48; Private; Assoc-adm; 12; Never-married; Craft-repair; Not-in-family; White; Male; United-States; <=50K> |
| R8, simh = 0.9219 | ¡62; Private; HS-grad; 9; Never-married; Craft-repair; Not-in-family; White; Male; United-States; <=50K> |
| R9, simh = 0.9141 | <28; Private; HS-grad; 9; Never-married; Craft-repair; Other-relative; White; Male; United-States; <=50K> |
| R10, simh = 0.9766 | <38; Private; HS-grad; 9; Divorced; Sales; Not-in-family; White; Male; United-States; <=50K> |
| R11, simh = 0.9688 | <70; Private; HS-grad; 9; Never-married; Craft-repair; Other-relative; White; Male; United-States; <=50K> |
| R12, simh = 0.9844 | <33; Private; HS-grad; 9; Divorced; Sales; Not-in-family; White; Male; United-States; <=50K> |

Не всі результати представлені в таблиці 2, але було 48842 профілів в базі даних. В наступних тестах було зроблено 48842 запити над тими ж 48842 профілями що привели до 2385540964 результатів для оцінки при вибраному рівню подібності. Загальна кількість результатів змінна і залежить від розміру бази даних та рівня подібності. Чим більший рівень подібності тим менша загальна кількість вилучених профілів.

В наступних підрозділах ми представимо Hamming DHT та the HCube – два різні шляхи розподілу запропонованої системи пошуку подібності.

### 4.1 Hamming DHT

Hamming DHT переймає від Chord [18] підхід послідовного хешування та процедури приєднання та залишення, але пропонує дві нові властивості націлені на отримання максимальної переваги від запропонованого механізму генерації подібних ідентифікаторів для подібного контенту. Коротко, цими двома новими властивостями є: 1) використання кодів Грея для організації ідентифікаторів в кільце; 2) встановлення вказівників на основі відстані за Хеммінгом ідентифікаторів вузлів.

Ця частина представляє аспекти зберігання та вилучення в запропонованій системі, де механізм класифікації контенту створений використовуючи RHH функцію, що відповідає наступним властивостям:

– ∀ c1,c2 ∈ C : simcos(c1,c2) → [0..1], де c1 та c2 – вектори контенту в просторі векторів контенту C.

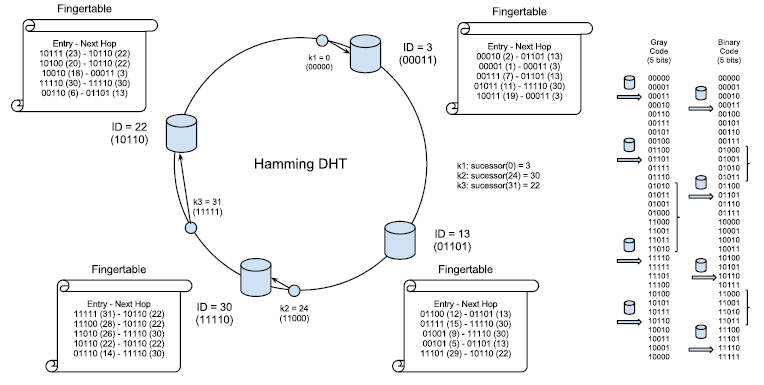
– ∀ c1,c2 ∈ C : Dh(RHH(c1),RHH(c2)) ∝ 1/simcos(c1,c2).

По суті, я показано в [19], властивості характерні для RHH функцій можуть представляти, з високою точністю, подібність між елементами контенту виміряну в подібності за Хеммінгом. Такі характеристики в поєднанні з організацією ідентифікаторів за кодами Грея в Hamming DHT та встановлення вказівників заснованому на відстані за Хеммінгом між вузлами, забезпечують ефективну систему пошуку подібності зменшуючи відстань (в стрибках) між вузлами що зберігають подібний контент.

#### 4.1.1. Побудова кільця

Всі вузли створюють віртуальне кільце HammingDHT та мають m-бітний ідентифікатор вузла. Щоб отримати такий m-бітний ідентифікатор вузли можуть використовувати будь-яку базову хеш-функцію як MD5 чи SHA-1, вирахувану над IP-адресою вузла та/або його приватним ключем, забезпечуючи унікальність ідентифікаторів.

Далі, після отримання своїх ідентифікаторів, вузли об’єднуються в кільце що організоване згідно послідовності коду Грея, на відміну від інших DHT як, наприклад, Chord, де кільце організоване півмісяцем в натуральному порядку ідентифікаторів. На мал. 4 показаний приклад запропонованої Hamming DHT з m=5. Як можна бачити на малюнку є чотири вузли (3 – 000112, 13 – 011012, 30 – 111102 та 22 – 101102) та три елементи (0 – 000002, 24 – 110002 та 31 – 111112). З правого боку мал. 4 можна бачити різницю між послідовністю коду Грея та серповидним натуральним порядком ідентифікаторів. З послідовності коду Грея маємо: 3 < 13 < 30 < 22.



Мал. 4. Приклад кільця Hamming DHT з m=5

#### 4.1.2. Послідовне хешування

Щоб зберігати контент в Hamming DHT, ключ k контенту призначається першому вузлу чий ідентифікатор більший чи рівний k згідно послідовності коду Грея. Цей вузол називається наступником ключа k і позначається successor(k). Коротко, successor(k) – це перший вузол за годинниковою стрілкою від k в m-бітному кільці коду Грея. Наприклад, вузол 3 відповідальний за зберігання контенту k=0, вузол 30 – за зберігання контенту k=24 та вузол 22 за – k=31.

Порівнюючи послідовність коду Грея та натуральний двійковий код (також показаний з правого боку на мал. 4) можна побачити деякі переваги послідовності коду Грея для об’єднання схожого контенту. Наприклад, всі елементи контенту з ідентифікаторами \*10\*\* послідовно розташовуються та зберігаються в вузлі 30 (11110), в той час як при натуральному двійковому порядку елементи зберігатимуться в двох різних місцях – вузлах 13 (01101) та 30 (11110).

#### 4.1.3. Встановлення вказівників

Коли кільце організоване, можна зберігати та вилучати інформацію на/з DHT якщо кожен вузол з’єднаний зі своїм наступником в кільці. На основі цих кругових взаємозв’язків дія зберігання чи вилучення даного ключа k вимагає маршрутизацію повідомлень по кільцю Грея через групу вузлів-наступників поки не буде знайдений вузол відповідальний за цей ключ контенту (successor(k)).

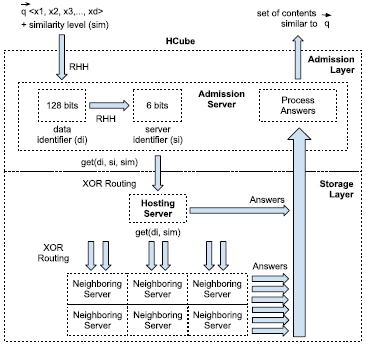
Однак для отримання кращої ефективності маршрутизації кожен вузол може містити таблицю маршрутизації, котру також називають таблицею вказівників, з (не більше ніж) m записами. Коли необхідний запис вказівника не вказує на вузол в DHT, він [запис] пов’язується з наступником.

i-ий запис (fi) в таблиці вказівників вузла p відповідає ідентифікатору отриманому заміною i-го біту ідентифікатору вузла. Запис (fi) пов’язується з вузлом successor(fi) в кільці коду Грея, тобто fi=(p⊕2i−1)→sucessor( fi), 1≤ i≤ m. Таблиця вказівників також містить локатор (наприклад, IP-адресу виділену вузлу) sucessor( fi). Наприклад, на мал. 4 показані таблиці вказівників для вузлів 3, 13, 30 та 22. В цьому прикладі таблиця вказівників вузла 13 вказує на наступника кожного з його m записів: 12, що вказує на 13; 15, що вказує на 30; 9 – на 30; 5 -- на 13 та 29 – на 22. Коли вузол p не має вказівника прямо установленого на successor(k) ключа k, він пересилає повідомлення вузлу  наявному в його таблиці вказівників, чий ідентифікатор передує k в кільці Грея. Цей процес повторюється поки не дійде до successor(k), відповідно до ситуації де число стрибків між вузлами більше ніж один.

Загальний опис архітектури HCube наведений в наступній частині, де показано організацію серверів в структурі дата центру і описано як ідентифікатори серверів назначаються згідно послідовності коду Грея та показано маршрутизацію eXclusive OR (XOR) відповідальну за пересилання трафіку.

### 4.2. HCube

Як можна бачити з мал. 5, логічна організація HCube складається з двох шарів – шару прийому та зберігання.



Мал. 5. Робота H-cube при запиті

Шар прийому забезпечує інтерфейс між зовнішнім світом та структурою HCube. Цей шар складається з набору серверів прийому котрі працюють зверху HCube, отримуючи запити від користувачів/застосувань та готуючи ці запити для введення в HCube для здійснення пошуку подібності. Для простоти, на мал. 5 показаний тільки один сервер прийому на шарі прийому.