УДК 004.023

Ю.Р. Піговський

ЕФЕКТИВНИЙ ЕМПІРИЧНИЙ МЕТОД ДЕДУБЛІКАЦІЇ НА ФАЙЛОВОМУ РІВНІ

Вдосконалено метод пошуку дублікатів контенту у файловій системі на основі емпіричного правила доцільності хешування. Правило створено на основі побудови математичних сподівань тривалості процедур хешування і попарного порівняння файлів. Проведено експериментальні дослідження методу.

Вступ

Дедублікація – це процес усунення дублікатних копій даних. Коли повторюваність (реплікованість) даних є доволі високою, що притаманне серверам резервного копіювання, образам віртуальних машин і репозиторіям сирцевих (source) кодів, дедублікація може зменшити витрати простору і збільшити швидкість обробки даних не те, що на проценти, а навіть на порядки [1].

Заощадження простору є очевидним наслідком дедублікації; підвищення ж швидкості відбувається внаслідок уникнення операцій запису на диск при збереженні дубльованих даних, а також зменшеного обсягу задіяної віртуальної пам'яті за рахунок спільного використання її сторінок багатьма застосунками.

В загальному випадку, дедублікація може розглядатися на одному з трьох рівнів: файловому, блоковому або байтовому. Дедублікація перших двох рівнів має практичний зміст, тоді як недоцільність дедублікації на байтовому рівні обгрунтовано в [1].

Відомим розв'язком задачі дедублікації на файловому рівні є утиліта A. Лопеза fdupes [2], що входить до складу GNU Coreutils. Проте алгоритм її роботи недостатньо ефективний, бо обчилює MD5 хеш кожного файла безумовно, не враховуючи доцільності такого обчислення для конкрентного співвідношення швидкості доступу до файлів та їхнього обсягу.

На блоковому рівні задачу дедублікації розв'язано Дж. Бонвіком [1] в модулі файлової системи ZFS. Дедублікація в ZFS має велику кількість допустимих комбінацій параметрів, зокрема: хешування алгоритмом SHA256, проста побайтова верифікація або хешування алгоритмом Флетчера (fletcher4) з побайтовою верифікацією. При цьому існує проблема у виборі найефективніших налаштувань, оскільки на деяких даних хешування алгоритмом SHA256 може відбуватися повільніше ніж комбінований підхід хешування Флетчера (fletcher4) з побайтовою верифікацією і навпаки.

Дану статтю присвячено підвищенню ефективності процедури дедублікації на файловому рівні шляхом побудови теоретичного правила, що дозволило б визначити при яких співвідношеннях швидкості доступу до файлів та їхнього обсягу доцільно проводити хешування, а при яких – ні.

Ефективність теоретичного правила буде перевірено експериментально.

Правило доцільності хешування

На сторінці вбудованої допомоги Linux (manpage) утиліту A. Лопеза описано так: "Searches the given path for duplicate files. Such files are found by comparing file sizes and MD5 signatures, followed by a byte-by-byte comparison" ("Шукає дублікати файлів у вказаній папці. Такі файли знаходяться шляхом порівняння розміру файлів та їхніх MD5 дайджестів, після чого слідує їх побайтове порівняння" [2]. Недоліком такого алгоритму є те, що цілком недоцільно обробляти незначну кількість файлів великого розміру хеш-функцією MD5 [3] перед їх попарним порівнянням. Обгрунтуємо це твердження теоретично.

Нехай N_L – кількість файлів, кожен з яких має розмір L байт. Хеш-функція –

це функція, що приймає на вхід рядок змінної довжини, який називають вхідним образом (контентом), а повертає рядок фіксованої (звичайно меншої) довжини – хеш (дайджест). Тривалість роботи цієї функції лінійно залежить від кількості файлів та їхньої довжини, тому її складність $O(N_L L)$. Збіг значень дайджеста є необхідною, проте недостатньою умовою еквівалентності контенту [3].

Обчислення дайджестів дає можливість розбити множину з N_L файлів на K_L груп з однаковим дайджестом (далі – дайджест-групи), в кожній з яких буде $n_{L,k}$ файлів, $N_L = \Sigma_{1 \le k \le K_L} n_{L,k}$.

Розбиття на дайджест-групи дасть змогу зекономити час на пошук дублікатів, за рахунок того, що тривалість хешування файлів разом з тривалістю їх попарного порівняння в K_L групах може бути меншою за тривалість попарного порівняння всіх N_L файлів.

Пояснимо це на екстремальному випадку, коли більшість файлів відрізняється вже першим блоком. Пошук дублікатів методом попарного порівняння в такому випадку буде особливо ефективний. Тривалість процедури попарного порівняння в основному визначатиметься тривалістю відкриття-закриття файлу t_{pr}, тобто підготовки внутрішніх структур даних ядра ОС при обслуговуванні файлової системи. Процедура попарного порівняння являє собою два цикли, вкладені один в другий, а тому має квадратичну складність $O(n^2)$ від кількості файлів n.

Навіть у таких, найсприятливіших для попарного порівняння умовах, може виявитися, що складність попарного порівняння великої кількості файлів $O(N_L^2)$ значно перевищить сумарну складність хешування $O(N_L L)$ і їхнього попарного порівняння у малих дайджест-групах $O\left(\Sigma_{1 \le k \le K_L} n_{L,k}^2\right)$. Крім того, попарне порівняння буде проводитися не у всіх групах, а лише в тих, до яких ввійшло два і більше файлів

$$\eta = \sum_{\substack{1 \le k \le K_L, \\ n_{L,k} \ge 2}} n_{L,k} \le N_L. \qquad (1)$$

Далі наведені оцінки складності алгоритмів будуть уточнені.

При $N_L = 2$ хешування, очевидно, марне, бо коли ці два файли матимуть різні дайджести, то їх хешування не відбудеться швидше ніж побайтове порівняння їхнього вмісту, навіть у найгіршому випадку, де вони відрізняються лише останнім байтом. Коли ж дайджести співпадуть, то до часу хешування додасться ще й час порівняння вмісту цієї пари файлів.

При $N_L \ge 3$ хешування, в деяких випадках, може бути і доцільним. Спробуємо знайти правило, що дозволить сказати чи для заданої кількості файлів N_L та довжини L доцільно проводити хешування.

Нехай $T_C(n, L)$ – тривалість попарного порівняння n файлів, кожен з яких має довжину L байтів. Для її оцінювання проаналізуємо алгоритм попарного порівняння файлів.

```
1. i ← 0
2. Поки i < n − 1 робити
3.     Якщо FILES[ i ] = Λ , то крок 24
4.     DUPLES ← Λ
5.     j ← i + 1
6.     fd1 ← open(FILES[ i ])
7.     Поки j < n робити
8.         Якщо FILES[ j ] = Λ , то крок 18
9.         fd2 ← open(FILES[ j ])
10.        Якщо cmpContent(fd1, fd2), то
11.            Якщо DUPLES = Λ , то
12.                DUPLES ⇐ FILES[ i ]
13.            Кінець якщо
14.            DUPLES ⇐ FILES[ j ]
15.            FILES[ j ] ← Λ
16.        Кінець якщо
17.        close (fd2)
18.        j ← j + 1
19.    Кінець поки
20.    close (fd1)
21.    Якщо DUPLES ≠ Λ , то
22.        ALLDUPLES ⇐ DUPLES
```

23. Кінець якщо
24. $i \leftarrow i+1$
25. Кінець поки.

При описі алгоритму використано нотацію Д. Кнута [4], де символ "Λ" означає порожнє посилання (порожній список), символ "$\Leftarrow$" означає занесення елемента, що знаходиться праворуч у список чи стек, що знаходиться ліворуч.

Операції "open" та "close" призначені для відкриття і закриття дескрипторів файлів (fd1, fd2), сума тривалостей їхнього виконання складає величину t_{pr}.

Нехай FILES – лінійний список шляхів до n файлів однакового розміру L, ALLDUPLES – список стеків з шляхами до файлів-дублікатів. На початку роботи він порожній: ALLDUPLES=Λ.

На кожній ітерації внутрішнього циклу (кроки 7–19) виконується наповнення локального стеку DUPLES дублікатами *i*-го файла. Після завершення цього циклу, непорожній локальний стек DUPLES добавляється у результуючий список дублікатів ALLDUPLES.

Тривалість роботи $T_C(n,L)$ алгоритму залежить від того, наскільки довго триватиме виконання кроку 10 на кожній ітерації попарного порівняння. На цьому кроці булева функція *cmpContent(fd1, fd2)*, в найпростішому випадку, виконує лінійний пошук відмінності контенту відкритих файлів fd1 та fd2 і зупиняється при знаходженні відмінного байта, повертаючи **false**, або не знаходить відмінності і повертає **true**. Зупинка може відбутися на будь-якому від першого до останнього з L байтів.

Припустимо, що номер першого байта, який відрізняє контенти файлів fd1 та fd2 є дискретною випадковою величиною m, що набуває значень у діапазоні $1 \le m \le L$, тоді $T_C(n,L)$ можна оцінити так:

$$T_C(n,L) = \\ = (n-1)t_{pr} + M(n)\cdot(t_{pr} + 2m\cdot t_r), \quad (2)$$

де $(n-1)$ – кількість разів, яку виконується крок 6, t_{pr} – тривалість приготувань, тобто середній час, що потрібен на відкриття і закриття одного файла, $M(n)$ – кількість разів, яку виконується крок 10, m – випадковий номер першого байта, що відрізняє контенти файлів fd1 та fd2, t_r – середня тривалість зчитування одного байта з жорсткого чи мережевого диска (теоретичний параметр, оскільки справжні реалізації алгоритму працюють з блоками). Слід зауважити, що у випадку еквівалентності контентів маємо $m = L$.

Кількість $M(n)$ разів, яку виконується крок 10 можна оцінити як суму арифметичної прогресії

$$M(n) = (n-1)+(n-2)+\cdots+1 = \\ = \frac{n}{2}(n-1). \quad (3)$$

Припустимо, що номер першого байта m, що відрізняє контенти файлів fd1 та fd2 є рівномірно-розподіленою в діапазоні $1 \le m \le L$ випадковою величиною з матсподіванням $\overline{m} = (1+L)/2$. Підставивши цю величину та (3) в (2) отримаємо таку оцінку математичного сподівання тривалості $T_C(n,L)$ виконання алгоритму:

$$\overline{T_C}(n,L) = (n-1)t_{pr} + \\ + \frac{n(n-1)}{2}\left(t_{pr} + (1+L)\cdot t_r\right). \quad (4)$$

З останнього співвідношення можна стверджувати, що алгоритм попарного порівняння n файлів має складність

$$O\left(n^2(t_{pr} + Lt_r)\right),$$

де n – кількість файлів, L – їхній осяг.

Беручи до уваги співвідношення (4) можна запропонувати наступне правило: хешування проводити доцільно, коли оцінка його тривалості менша за математичне сподівання тривалості попарного порівняння

$$H(n,L) = \begin{cases} \text{так}, & T_H(n,L) < \overline{T_C}(n,L); \\ \text{ні}, & \text{інакше}, \end{cases} \quad (5)$$

де $T_H(n,L)$ – тривалість обробки n файлів обсягом L байт деякою хеш-функцією (хешування).

Відомо велику кількість функцій хешування. З перспективи поставленої у статті задачі, їхніми найважливішими характеристиками є швидкість обчислення та імовірність колізій.

Розглянемо кілька альтернативних функцій хешування на предмет того, котра з них краще підходить для розв'язання поставленої у статті задачі.

Хеш-функція SHA-1 має широке прикладне застосування для задачі індексування версій файла в сучасних системах контролю версій, таких як Git, Mercurial та Monotone [5]. Вона відзначається надзвичайно малою імовірністю колізії, завдяки досить великій довжині дайджеста (20 байт) і лавиноподібності його зміни при незначній зміні контенту. Проте має складний алгоритм роботи.

Іншим варіантом є функція хешування на основі алгоритму обгляду нелінійної таблиці (NTL – Nonlinear Table Lookup). Вона відзначається високою швидкодією і простотою реалізації, проте, у порівнянні з SHA-1, вразливіша до імовірних колізій [3].

Хеш-функція MD5, використана А. Лопезом [2], посідає проміжне місце між NTL та SHA-1 за швидкодією та імовірністю колізії. Вона не настільки швидкісна як NTL, але й стійкіша, у порівнянні з ним, до колізій, хоча й поступається надійністю алгоритму SHA.

Усі ці алгоритми мають лінійну складність, тому тривалість їхньої роботи $T_H(n,L)$ лінійно залежить від обсягу оброблюваних ними файлів

$$T_H(n,L) = n(t_{ph} + L \cdot t_h) + M(n)t_c, \quad (6)$$

де n – кількість файлів довжиною L байтів, t_{ph} – середній час, що витрачається на підготовку до хешування незалежно від об'єму оброблюваного файлу, t_h – час, що в середньому припадає на хешування одного байту, t_c – тривалість попарного порівняння дайджестів для визначення до якої дайджест-групи віднести щойно прохешований файл.

Оскільки операції попарного порівняння дайджестів виконуються в оперативній пам'яті комп'ютера, то час t_c є настільки мізерним, що складність алгоритму хешування можна оцінити як $O(n \cdot Lt_h)$.

В наступному розділі буде проведено експериментальне дослідження побудованого правила.

Бенчмаркінг введення-виведення

Чисельні експерименти проведено на жорсткому диску з файловою системою NTFS, корисний обсяг якого складає 465 Гб, 234 Гб з яких використовуються 634 589 файлами (загальним обсягом 226 Гб) в 95 189 папках.

Пошук файлів на жорсткому диску виявив 46 150 груп файлів однакового розміру. Загальна кількість файлів, що створюють групи з двох і більше файлів, згідно з формулою (1), складає $\eta = 599\,368$, тобто 94% від загальної їх кількості.

Найчисельнішою групою файлів однакового розміру виявилася група з 5110 файлів, що мають розмір 8 Кб (8192 байт). Потім виявлено 4 678 порожніх файлів. Присутність такої кількості порожніх файлів можна пояснити тим, що деяке програмне забезпечення використовує порожні файли, як прапорці, що керують включенням або виключенням тих чи інших функцій ПЗ (наприклад, це характерно для деяких CMS), користувачі-розробники часто створюють порожні файли-стаби, які планують наповнити вмістом згодом, а також тим, що антивірусні програми усувають код вірусів з виконуваних файлів, після чого вони перетворюються на порожні іменовані файли.

Серед лідерів за чисельністю також можна назвати групи з 3 784 файлів довжиною 402 байти, 3 186 – довжиною 53 байти та 2 892 – довжиною 2 байти.

На початку дослідження проаналізуємо, яку кількість з виявлених 46 150 груп файлів однакового розміру слід обробити хеш-функцією. Для цього потрібно обчислити для кожної групи вираз (5) за допомогою формул (4) та (6). Таке обчислення можливе лише за умови відомості коефіцієнтів t_{ph}, t_{pr}, t_h, t_r та t_c. Їх можна отримати за допомогою так зва-

ного бенчмаркінгу (Benchmarking) або на основі емпіричних міркувань.

Процедури бенчмаркінгу були побудовані таким чином, щоб можна було оцінити можливість наближення тривалості процедур зчитування інформації та хешування лінійними функціями.

Тривалість хешування вимірювалася простим фіксуванням різниці часу між початком і завершенням процедури хешування. Як вже було сказано раніше, тривалість t_c попарного порівняння дайджестів для визначення до якої дайджест-групи віднести щойно прохешований файл – дуже незначна, оскільки таке порівняння не вимагає роботи із зовнішніми пристроями пам'яті, а всі порівняння проводяться безпосередньо в оперативній пам'яті. Вимірювання такої незначної тривалості дуже утруднене і може містити значні метрологічні похибки, тому у формулах покладемо $t_c = 0$, але неявно включатимемо її при оцінці t_{ph}.

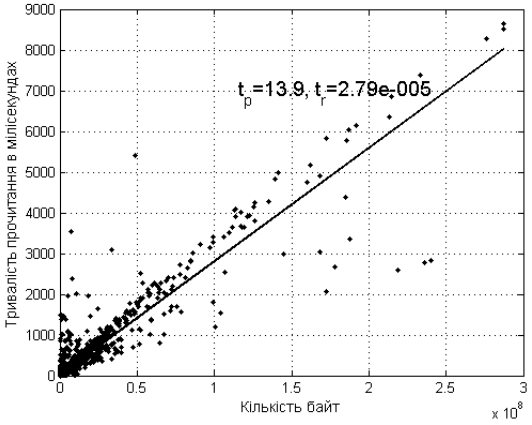
Тривалість зчитування інформації вимірювалася при виконанні процедури попарного порівняння до появи першої відмінності між файлами. Таким чином нагромаджувалися дані про тривалість зчитування різної кількості інформації одразу з двох файлів, що слід врахувати при аналізі результатів.

Підсумкова тривалість при співпадінні обсягу прочитаної інформації усереднювалася за формулою

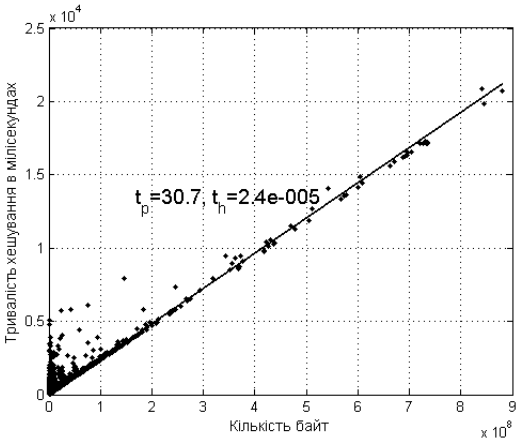
$$t_{L,k+1} = \left(t_{L,k} + t_{L,k-1}\right)/2,$$

де $t_{L,k}$ – поточна оцінка тривалості зчитування L байт інформації, $t_{L,k-1}$ – попередня оцінка цієї тривалості.

Результати бенчмаркінгу, на рис. 1, показують, що тривалість підготовки до роботи з файлами $t_{ph} \approx t_{pr} \approx 10^2$ мілісекунд на сім порядків перевищує тривалість обробки одного байта як у процедурі зчитування, так і в процедурі хешування $t_h \approx t_r \approx 10^{-5}$ мс. Це пов'язано з тим, що при відкритті файла на жорсткому диску відбувається створення та ініціалізація внутрішніх структур даних ОС для дескрипторів, створення і наповнення буферів введення-виведення, а при відкритті файла з мережевого файлового сховища відбувається встановлення з'єднання, виконання так званого протоколу рукостискання, аутентифікації та авторизації. Водночас t_r включає лише послідовне прочитання даних з носія або мережевого ресурсу після виконання всіх підготовчих дій, що, зазвичай, оптимізується алгоритмами випереджуючого читання в контролерах сучасних жорстких дисків, кешування в ОС та протоколах мережевої взаємодії.



а



б

Рис. 1. Спостереження тривалості зчитування (а) та хешування (б) файлів різного обсягу

При огляді графіків помітні такі два явища: середня тривалість хешування одного байта $t_h = 2.40 \cdot 10^{-5}$ мс. виявилася, хоча і несуттєво, але меншою від тривалості простого зчитування $t_r = 2.79 \cdot 10^{-5}$ мс., оскільки одночасне читання з двох різних

файлів утруднює роботу алгоритму випереджаючого читання, що працює безперешкодно при хешуванні одного файлу. Відносна дисперсія спостережень тривалості зчитування файлів більша за відносну дисперсію тривалості хешування. Ці явища пояснюються тим, що у випадку з хешуванням програма обробляє на кожній ітерації єдиний, причому, завжди інший файл, що не дозволяє ОС застосувати кеш, тоді як, у випадку з попарним порівнянням, дисперсія вниз від лінії тренду означає зменшення тривалості зчитування за рахунок дії кешу ОС для файлів, що відкриваються вже не вперше. Дисперсія вгору від лінії тренду пояснюється фрагментацією файлової системи та затримками, що спричинені діями інших задач ОС.

Отримані оцінки бенчмаркінгу узгоджуються з даними, що наведені в літературі, зокрема, в [6], де розроблено та досліджено ПЗ Parabench оцінювання продуктивності зберігаючих пристроїв та мереж (Storage Area Network) шляхом імітації функціонування застосунків, що зчитують і зберігають великі обсяги даних.

Експериментальні дослідження

Експерименти показали, що пошук дублікатів серед 634 589 файлів алгоритмом побайтового порівняння без хешування на описаному в попередньому розділі жорсткому диску триває 16 862.3 сек. (4 год. 41 хв.), тривалість роботи алгоритму хешування-побайтового порівняння не надто відрізняється – 16 405 сек. (4 год. 33 хв.).

Проте аналіз в розрізі груп файлів однакового розміру показав теоретичну можливість скоротити цей час до 14 816 сек., тобто на 12.12 % щодо алгоритму без хешування чи 9.69 % щодо алгоритму з хешуванням.

Такий незначний, на перший погляд, виграш часу, може обернутися суттєвим виграшем при зростанні обсягу, кількості файлів і зберігальних пристроїв у мережах Storage Area Networks.

На графіку рис. 2 порівняно ефективність роботи двох алгоритмів: без хешування (пунктирна крива) та з хешуванням (суцільна крива). Логарифмічна вісь ординат показує кількість сек., що знадобилася на знаходження дублікатів серед тієї кількості файлів, що подана віссю абсцис. Для групи восьмикілобайтових (8192 байт) файлів, що налічує максимальну кількість з 5110 файлів (останні точки за віссю абсцис) помітно, що тривалість пошуку з хешуванням-побайтовим порівнянням є на порядок меншою від тривалості пошуку простим побайтовим порівнянням (60.3 сек. проти 621.0 сек.). Така разюча відмінність тривалості свідчить про низьку частоту колізій у даній групі, а отже велику ефективність алгоритму хешування в цьому випадку.

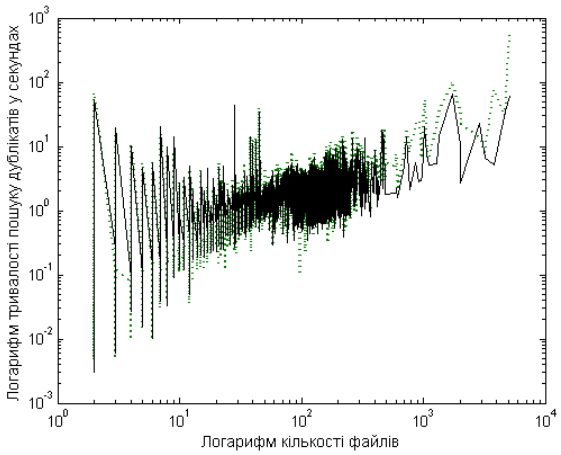


Рис. 2. Порівняння ефективності роботи алгоритмів без хешування та з хешуванням

Використовуючи отримані наближення коефіцієнтів, розрахунок виразу (5) за допомогою формул (4) та (6), показав, що хешування доцільно проводити лише для 5 373 груп файлів однакового розміру, тобто лише для 12 % від загальної кількості з 46 150 груп. При цьому виявилося також, що хешування доцільно проводити вже для груп, у яких кількість файлів перевищує 20.

Результати свідчать про те, що дедублікація із застосуванням співвідношення (5) триває 16 460 сек., що дає несуттєвий виграш швидкодії у порівнянні з алгоритмом без хешування і навіть дещо програє у швидкодії алгоритму з хешуванням.

З метою покращення результатів було узагальнено співвідношення (5) шляхом введення емпіричного коефіцієнту α:

$$H_\alpha(n,L)=\begin{cases}\text{так}, & T_H(n,L) < \alpha\overline{T_C}(n,L);\\ \text{ні}, & \text{інакше.}\end{cases} \quad (7)$$

Роль коефіцієнту α можна пояснити як засіб урівноваження припущень щодо розподілу випадкової величини позиції відмінності при попарному порівнянні файлів.

При $\alpha = 0.0$ алгоритм зведеться до алгоритму без хешування, при $\alpha = 1.0$ матимемо класичну версію (5), а для деякого достатньо великого $\alpha > 1.0$ алгоритм зведеться до алгоритму із безумовним хешуванням. В ході експериментів виявилося, що останній випадок настає вже при $\alpha > 1.8$.

На рис. 3 показано тривалість роботи узагальненого алгоритму з співвідношенням (7) для різних значень емпіричного коефіцієнту в діапазоні $0.0 \le \alpha \le 2.0$.

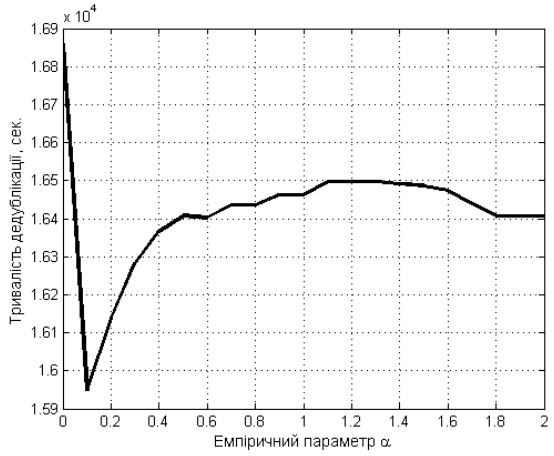


Рис. 3. Тривалість роботи узагальненого алгоритму для $0.0 \le \alpha \le 2.0$

З рисунку можна зробити висновок, що мінімальна тривалість дедублікації досягається при $\alpha = 0.1$ і складає 15 950 сек., що на 3 % швидше від алгоритму із безумовним хешуванням і на 5 % швидше від алгоритму без хешування.

Висновки

Створено правило доцільності виконання процедури хешування в залежності від кількості, обсягу та швидкості доступу до файлів у задачі пошуку дублікатів контенту.

Правило створено на основі теоретичних досліджень математичного сподівання тривалості процедур хешування та попарного порівняння.

Розроблене правило дає змогу знайти ті групи файлів однакового розміру, в котрих доцільно проводити хешування з метою підвищення ефективності пошуку дублікатів на файловому рівні. Дедублікація розробленим методом триває 15 950 сек., що на 3 % швидше від алгоритму із безумовним хешуванням і на 5 % швидше від алгоритму без хешування.

У наступних дослідженнях планується покращити розроблений метод з метою збільшення виграшу швидкодії і поширити область його застосування на задачу дедублікації на блоковому рівні.

1. *Bonwick J.* ZFS Deduplication. – 2009 [Електронний ресурс]. – Режим доступу: https://blogs.oracle.com/bonwick/entry/zfs_dedup
2. *Lopez A.* fdupes(1) – Linux man page [Електронний ресурс]. – Режим доступу: http://linux.die.net/man/1/fdupes
3. *Шеховцов В.А.* Операційні системи // Захист інформації в операційних системах. – К.: Видавнича група BHV, 2005. – Розд. 18. – С. 471–472.
4. *Кнут Д.Э.* Искусство программирования, том 1. Основные алгоритмы, 3-е изд.: Пер. с англ.: Уч. пос. – М.: Издательский дом "Вильямс", 2000. – 720 с.
5. *Chacon S.* Pro Git. – Apress, 2009.
6. *Mordvinova O.* I/O Benchmarking of Data Intensive Applications / Olga Mordvinova, Thomas Ludwig, Christian Bartholomä // Проблеми програмування. – К.; 2010. – N 2–3. Спец. випуск. – С. 107–115.

Одержано 16.05.2014

Про автора:

Піговський Юрій Романович,
доцент кафедри комп'ютерних наук,
кандидат технічних наук,

Місце роботи автора:

Тернопільський національний
економічний університет,
46001, Тернопіль,
вул. Чехова, 8.
Тел.: 55 0971.
E-mail: pigovsky@gmail.com