УДК 004.75

О.В. Кузнецов[1], Л.Е. Чала[2], С.Г. Удовенко[3]
[1] ХНУРЕ, м. Харків, Україна, oleksii.kuznetsov@nure.ua
[2] ХНУРЕ, м. Харків, Україна , larysa.chala@nure.ua
[3] ХНЕУ ім. С.Кузнеця, м. Харків, Україна , serhii.udovenko@nure.ua

НЕЙРОМЕРЕЖЕВИЙ МЕТОД КЕШУВАННЯ ДАНИХ

Розглянуто основні існуючі види кешування та алгоритми збереження даних в кеш-памяті. Запропоновано підхід до здійснення кешування на основі нейронних мереж. Проаналізовано результати роботи проекту-доказу концепції. Запропонований метод кешування розглянуто як вирішення проблеми алгоритму Беладі. Визначено перспективи використання нейромережевого методу кешування.

НЕЙРОННА МЕРЕЖА, КЕШ, МЕТОД КЕШУВАННЯ, АЛГОРИТМ БЕЛАДІ, ПРОГНОЗУВАННЯ ЗАПИТІВ

А.В. Кузнецов, Л.Э. Чалая, С.Г. Удовенко. Нейросетевой метод кэширования данных. Рассмотрены основные существующие виды кэширования и алгоритмы сохранения данных в кэш-памяти. Предложен новый вид кэширования на основе нейронных сетей. Проанализированы результаты работы проекта-доказательства концепта. Предложенный метод кэширования рассмотрен как решение проблемы алгоритма Белади. Определены перспективы использования нейросетевого метода кеширования.

НЕЙРОННАЯ СЕТЬ, КЭШ, МЕТОД КЭШИРОВАНИЯ, АЛГОРИТМ БЕЛАДИ, ПРОГНОЗИРОВАНИЕ ЗАПРОСОВ

O.V. Kuznetsov, L.E. Chala, S.G. Udovenko. Neural network data caching method. Main existing types of caching and algorithms of saving cached data were analyzed. The new type of caching based on neural networks was proposed. Results of proof of concept project were analyzed. Proposed type of caching was reviewed as a solution to a problem of Belady algorithm. The scale of subject area to use neural type of caching was determined.

NEURAL NETWORK, CACHE, TYPE OF CACHING, BELADY ALGORITHM, QUERY PREDICTION

Вступ

Одним із найбільш поширених способів оптимізації web-систем є кешування даних [1]. Дані, що знаходяться далеко від користувача, потребують значного часу на їх передачу. В більшості сучасних програм використовуються зустрічаються повільні операції, результати яких можна зберегти на деякий час. Це дозволяє оперативно видавати користувачам заздалегідь збережені дані. Основна мета кешування для веб-серверів – зменшення часу на отримання контенту, аби користувач якомога швидше отримував необхідні дані. Тому кешування, як відносно просту технологію, часто впроваджують в браузерах та проксі-серверах інформаційних систем.

Відзначимо, що суттєвий час витрачається на обробку даних системи зі сторони постачальника та передачу їх до клієнта. За цих умов, при наявності великих навантажень на канали системи кешування є особливо корисним. Воно дозволяє, зазвичай, обслуговувати більше клієнтів без додаткового збільшення ресурсів за рахунок скорочення навантажень на основні джерела даних. Втім, навіть при невеликих навантаженнях кешування може дуже позитивно вплинути на швидкість роботи с інформаційною системою.

Важливою проблемою забезпечення ефективності методів кешування є врахування фактору старіння даних. Якщо не перевіряти механізм зберігання закешованих даних, користувач може довгий час не отримувати актуальні дані, внаслідок чого можуть блокуватися корисні потоки даних для багатьох клієнтів.

Тож дуже важливо розуміти, які саме дані є доцільно кешувати, щоб не зруйнувати логіку роботи системи. При цьому необхідно визначити, наскільки актуальними мають бути керовані дані. Враховуючи стрімке зростання потужностей, що використовуються для підтримки роботи інформаційних систем, доцільним є залучення до вирішення цього питання нейромережевих засобів.

У даній роботі досліджуються сучасні методи кешування та зберігання даних, а також пропонується нейромережевий підхід до реалізації процесу кешування з використанням алгоритму Беладі.

1. Аналіз існуючих видів кешування

Розглянемо три основних типи кешування даних, що використовуються в інформаційних системах [1, 2].

Lazy cache («ледачий» кеш) – найпростіший в реалізації тип кешування, що часто вбудовується в фреймворки. Такий кеш просто зберігає дані і віддає їх, поки вони не застаріють.

Synchronized cache (синхронізований кеш) – клієнт разом з даними аналізує мітку останньої зміни і може запитувати у постачальника, чи не змінилися дані, щоб повторно це не запитувати. Такий тип кешування дозволяє завжди мати актуальні дані, але є дещо складним в реалізації.

Write-through cache (кеш наскрізного запису) – будь-яка зміна даних відображається відразу

в сховищі і в кеші. Цей тип кешу може ніколи не застарівати, але при цьому виникають проблеми з так званої когерентності процесу.

Обсяг кешу завжди обмежений. Найчастіше він менше обсягу даних, які в цей кеш можуть надходити, тому елементи, поміщені до кешу, рано чи пізно будуть замінені. Сучасні фреймворки для кешування дозволяють достатньо гнучко управляти старінням, з оглядом на пріоритети, час старіння, обсяги даних тощо.

Якщо одні й ті ж дані потрапляють в різні кеши, то виникає проблема когерентності кешу. Наприклад, одні й ті ж дані можуть використовуватися для формування різних сторінок їх кешування. Сторінки, які сформовані пізніше, будуть містити оновлені дані, а сторінки, що кешувалися раніше, будуть містити застарілі дані. В цьому разі буде порушена узгодженість поведінки.

Простий спосіб підтримки когерентності - примусове старіння (скидання) кешу при зміні даних. Тому збільшення пам'яті для кешу, щоб він менше застарівав, не завжди є доцільним.

Основний параметр, який характеризує систему кешування – це відсоток влучень запитів в кеш. Цей параметр досить легко виміряти, щоб зрозуміти наскільки система кешування ефективна. Часті скиди кешу, кешування рідко запитуваних даних, недостатній обсяг призводять до марної трати оперативної (зазвичай) пам'яті, не підвищуючи ефективність роботи.

Іноді дані змінюються настільки часто і непередбачувано, що кешування не дає ефекту, а відсоток влучень буде близький до нуля. Але зазвичай дані зчитуються набагато частіше, ніж записуються, тоді кеши є ефективними.

Ледачий кеш – це найпростіший вид кешування, але його потрібно використовувати обережно, так як він часто дає застарілі дані. Можна при кожному записі скидати ледачий кеш, щоб підтримувати актуальність даних, але тоді витрати на реалізацію будуть високими в порівнянні з більш складними типами кешування.

На рис. 1. наведено псевдокод одного з можливих варіантів реалізації ледачого кешу.

```
getElement(elementId) {
    if (isElementAlreadyPresent(elementId)) {
        element = loadFromCache(elementId)
        if (element.isOld()) {
            element = loadFromDB(elementId)
            saveToCache(element, currentTime)
        }
    } else {
        element = loadFromDB(elementId)
        saveToCache(element, currentTime)
    }
    return element
}
```

Рис. 1. Псевдокод реалізації ледачого кешу

Такий тип кешування можна використовувати для даних, які майже ніколи не змінюються. Інший варіант його використання – створити ледачий кеш з невеликим часом старіння для стабільної роботи при сплесках навантаження. Такий тип ледачого кешування дозволить швидше давати відповідь при запитах.

Синхронізований кеш – це найкорисніший тип кешування, так як віддає свіжі дані і дозволяє реалізувати багаторівневий кеш. Такий тип кешування вбудований, зокрема, в протокол HTTP. Сервер віддає мітку зміни, а клієнт кешує у себе результат і в подальшому запиті передає цю мітку. Сервер може дати відповідь, що стан не змінився і можна використовувати кеш на клієнтному об'єкті. Сервер, в свою чергу, після отримання мітки може перепитати у сховища, були зміни чи ні. На рис. 2. наведено псевдокод одного з можливих варіантів реалізації синхронізованого кешу.

```
getElement(elementId, knownStateHash) {
    if (isElementAlreadyPresent(elementId)) {
        element = loadFromCache(elementId)
        if (element.getHash() != knownStateHas
            element = loadFromDB(elementId)
            saveToCache(element)
        }
    } else {
        element = loadFromDB(elementId)
        saveToCache(element)
    }
    return element
}
```

Рис. 2. Псевдокод реалізації синхронізованого кешу

Цей тип кешування не рятує від значних витрат на спілкування між системами. Тому часто він доповнюється іншими типами кешування, щоб прискорити роботу.

Розглянемо особливості кешу наскрізного запису. Його доцільно використовувати в системах розподіленого кешування (memcached, Windows Sever App Fabric, Azure Cache). Ручна реалізація процесу синхронізації кешів між вузлами є дуже затратною, тому її зазвичай не застосовують в рамках розробки програм. На рис. 3. наведено псевдокод одного з можливих варіантів реалізації наскрізного кешу.

```
getElement(elementId) {
    element = loadFromCache(elementId)
    return element
}

changeElement(elementId) {
    element = loadFromDB(elementId)
    performChanging(element)
    saveToDB(element)
    saveToCache(element)
}
```

Рис. 3. Псевдокод реалізації наскрізного кешу

Не варто намагатися кешувати всі дані в синхронізованому кеші, інакше велика частина коду програми буде займатися перебудовою кешу. Також не варто забувати, що системи розподіленого кешування також вимагають спілкування між системами, що може позначатися на швидкодії обміну даними.

Надзвичайно важливо обрати правильну гранулярність даних, що кешуються. Наприклад, кешування даних для кожного користувача швидше за все буде неефективним при великій кількості користувачів. Якщо кешувати дані для всіх користувачів разом, то можуть виникати проблеми зі старінням даних і когерентністю кешу.

Зазвичай дані кешуються дані зазвичай безпосередньо перед надсиланням в зовнішню систему. Кешувати дані, які були отримані ззовні, доцільно тільки в разі проблем з продуктивністю на цьому етапі. Зовнішні сховища, такі як СУБД і файлові системи, самі реалізують процес кешування.

2. Особливості застосування нейронних мереж для прогнозування запитів

На даний момент існує велика кількість задач, які неможливо вирішити без наявності великих потужностей, тому що для їх обробки необхідно мати великий обсяг пам'яті, а не лише процесорний час. Саме тому у сучасному світі існують і створюються спеціалізовані центри даних. Більше того, деякі з цих задач необхідно вирішувати з безпосередньою участю людини, тому питання миттєвого опрацювання запитів підіймається дуже часто.

Існує чимало систем, які використовують аналітику, основану на великих об'ємах даних. До них можна віднести як інформаційні системи для широкого кола користувачів (сайт мобільного оператора, систему пошуку квитків тощо), так і внутрішні корпоративні системи. Можна стверджувати, що користувачі таких систем на очікування результату запиту завжди витрачають більше часу, ніж для створення цього запиту. Що ж стосується самих запитів, то їх критерії часто повторюються. Деякі критерії можна спрогнозувати (вибірку за поточний місяць, вибірку серед усіх даних, доступних лише конкретному користувачеві тощо).

Зважаючи на усі передбачувані параметри, можна спрогнозувати запит за деякий час до того, як він буде складений, і зберегти результат його виконання. Коли користувач захоче отримати результат цього запиту, системі не потрібно буде виконувати його ще раз, аже результат можна буде отримати миттєво з пам'яті.

Слід відзначити, що задачу прогнозування можливих запитів дуже зручно розглядати з оглядом на можливість використання для цього нейронних мереж. Важко знайти сучасну інформаційну систему з великим об'ємом даних, яка б не збирала статистику по користувачам і їх діям. Зазвичай це допомагає мати краще уявлення про те, як клієнти користуються системою, які саме запити складаються найчастіше і де користувачі витрачають найбільше часу, щоб прискорити популярні частини системи. Для нейронної мережі це означає, що даних для навчальної вибірки завжди буде вдосталь. Більше того, вибірка буде поповнюватися з кожним днем і можна буде легко налаштувати нейромережу на основі реальних результатів її роботи.

Звичайно, навчання нейронної мережі, та її робота вимагає чимало ресурсів, але для крупних аналітичних систем такі витрати є цілком адекватними.

Розглянемо більш детально завдання для прогнозування, формування навчальної вибірки та визначення необхідних ресурсів.

Відбір даних для навчальної вибірки залежить, насамперед, від типу і напрямку інформаційної системи. Визначимо загальні правила формування навчальної вибірки.

Вихідні дані прогнозування формуються за допомогою набору фільтрів у запиті. Наприклад, для сайту мобільного оператора це може бути запит на виписку по грошовим витратам конкретного рахунку. Тут фільтрація буде виконана по ідентифікатору користувача, що виконав запит та часовому інтервалу результатів запиту. Для користувача корпораційної аналітичної системи вихідними даними буде набір фільтрів, що користувач власноруч відправив на обробку в систему.

Розглядаючи можливості оптимізації процесу зберігання результатів запиту, можна також брати до уваги час виклику запиту до вихідних даних, щоб виконати запит лише безпосередньо перед тим, як він знадобиться і не тримати результат запиту у пам'яті протягом деякого часу. Для багатьох корпоративних систем час виклику запиту буде достатнім для прогнозування часу його виклику наступного разу. Але існують варіанти, в яких ключову роль відіграє не час виклику, а стан бази даних. Так, наприклад, оператору порту важливо отримувати інформацію про стан складів після того, як він отримав повідомлення про закінчення розгрузки вантажного човна. Як відомо, час розгрузки, так само як і час прибуття, завжди різний і може відрізнятися не тільки годинами, а й днями.

Вхідні дані можуть кардинально відрізнятися в залежності від напрямку інформаційної системи. Для промислових інформаційних систем дуже важливо враховувати специфіку підприємства та загальний напрям його роботи. Працюючи з системою, що аналізує та підсумовує фінансові операції, логічно було б сказати, що на вхід нейронної мережі необхідно додати дані про користувача та його предметну область. Так можна опрацювати сценарій, в якому кожен клієнт інформаційної

системи відповідає за певну частину її даних. Тож маючи свідчення про те, що зазвичай запитує користувач, можна буде заздалегідь підготувати відповідь. Питання використання ресурсів є також відкритим. Архітектура системи та динаміка її використання значно впливають на можливі сценарії залучення нейронної мережі. Якщо на вході маємо систему, якою користуються багато клієнтів з географічно віддалених регіонів, то вона повинна бути доступна цілодобово. Це означає, що нейромережа має постійно навчатися і відповідати на запити. Якщо ж ми маємо систему, що працює з великими даними лише деяку частину дня, то доцільно використовувати час бездіяльності для прогнозування запитів на наступний бізнес-день. Таким чином, навантаження на систему буде рівномірним протягом доби і необхідність у виділенні додаткових ресурсів значно зменшиться. На рис. 4. наведено псевдокод одного з можливих варіантів реалізації нейромережевого кешу.

```
getElement(elementId) {
    if (isElementAlreadyPresent(elementId)) {
        element = loadFromCache(elementId)
    } else {
        element = loadFromDB(elementId)
        saveToCache(element, currentTime)
    }
    return element
}

analyzeElement(elementId) {
    possible = countPossibilitytoBeRequired(el
    if (possible) {
```

Рис. 4. Псевдокод реалізації нейромережевого кешу

Таким чином, існує можливість реалізації нового виду кешу – нейромережевого. Розгдянемо далі сучасні алгоритми зберігання керованих даних у пам'яті.

3. Аналіз існуючих алгоритмів збереження кешованих даних

Найбільш ефективне правило кешового оновлення – видаляти з кешу ту інформацію, яка не знадобиться в майбутньому найдовше. Відповідне оптимальне кешування можна реалізувати за допомогою алгоритму Беладі або алгоритму передбачення [1]. У загальному випадку неможливо точно передбачити, коли саме в наступний раз буде потрібна саме та інформація, що видаляється, тому на практиці реалізація такого передбачення є дуже складною. Параметри прогнозування можуть бути обчислені лише експериментальним шляхом, після чого можна запропонувати модифікацію алгоритму кешування з прогнозуванням та порівняти з ним ефективність поточного алгоритму кешування.

Згідно з алгоритмом Least recently used (LRU) в першу чергу з кешу витісняються дані, що не використовувалися найдовше [2]. Цей алгоритм потребує відстеження динаміки використання даних, що вимагає чимало часу, особливо якщо потрібно проводити додаткову перевірку відповідної статистики. Загальна реалізація цього методу вимагає збереження контрольних бітів «віку» для рядків кешу, за рахунок чого відбувається відстеження найменш використаних рядків (тобто за рахунок порівняння таких бітів). У цьому алгоритмі при кожному зверненні до рядка кешу змінюється «вік» всіх інших рядків (рис. 5).

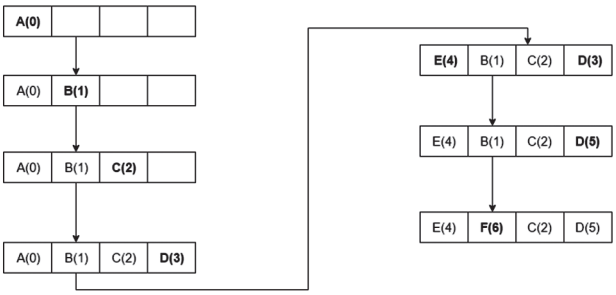

Рис. 5. Схема роботи LRU алгоритму

Згідно з алгоритмом Most Recently Used (MRU), на відміну від LRU, в першу чергу витісняється останній використаний елемент (рис. 6) [1]. Коли файл періодично сканується по циклічною схемою, то MRU є найкращим алгоритмом витіснення. Для схем довільного доступу і циклічного сканування великих наборів даних (іноді званих схемами циклічного доступу) алгоритми кешування MRU мають більше влучень в порівнянні з LRU за рахунок їх прагнення до збереження старих даних. Алгоритми MRU найбільш корисні у випадках, коли відбувається більше звернень до найстаріших елементів.

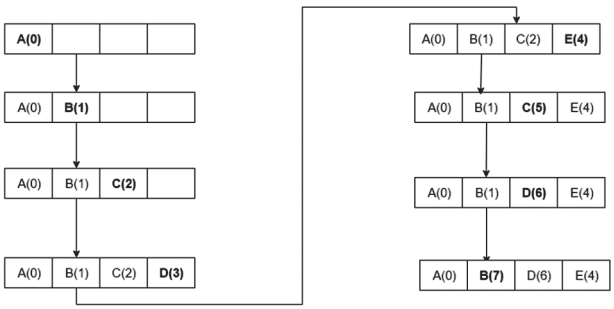

Рис. 6. Схема роботи MRU алгоритму

Для кешів з великою асоціативністю, ціна реалізації алгоритмів LRU стає непомірно високою. Якщо майже завжди потрібно відкидати найменш використовуваний елемент, то в цьому випадку більш ефективним є алгоритм PLRU (псевдо-LRU), що вимагає для елементів кешу тільки один додатковий біт (рис. 7) [3].

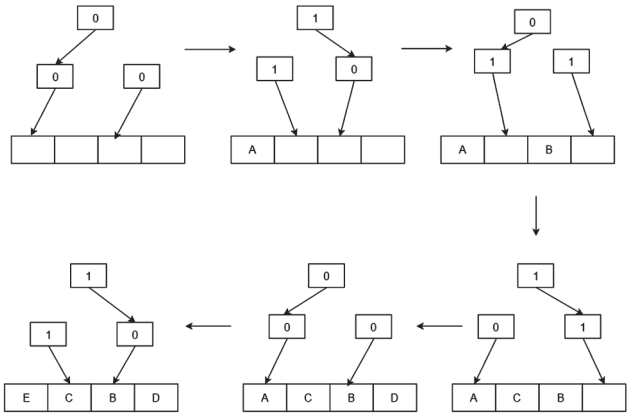

Рис. 7. Схема роботи PLRU алгоритму

Згідно з алгоритмом сегментованого LRU (Segmented LRU або SLRU) SLRU-кеш ділиться на пробний та захищений сегменти (рис. 8) [4]. Рядки в кожному сегменті впорядковані від часто використовуваних до найменш використовуваних. Дані в разі невлучень додаються в область останніх використаних елементів пробного сегмента кешу. Дані при влучанні видаляються (незалежно від місця їх розташування) і додаються в область часто використовуваних елементів захищеного сегменту. Таким чином, звернення до рядків захищеного сегменту відбуваються принаймні двічі. Перенесення рядку з пробного сегменту в захищений сегмент може викликати перенесення останнього використаного (LRU) рядку в захищеному сегменті в MRU-область пробного сегменту, даючи цієї лінії другий шанс бути використаною перед витісненням. Розмір захищеного сегмента є важливим SLRU-параметром, який змінюється в залежності від схеми роботи введення-виведення. Кожного разу, коли дані повинні бути витіснені з кешу, рядки запитуються з LRU-кінця пробного сегменту.

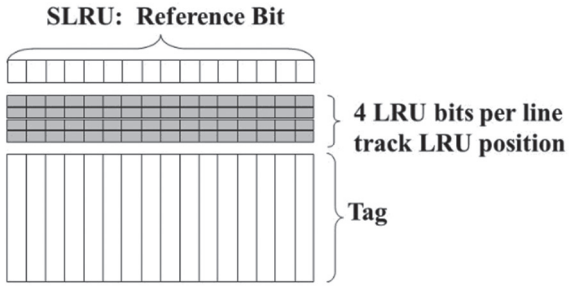

Рис. 8. Схема роботи SLRU алгоритму

Для високошвидкісного процесорного кешу, де навіть PLRU є занадто повільною, застосовується 2-канальна асоціативність (2-Way Set Associative) (рис. 9). Адреса нового елементу використовується для обчислення одного з двох можливих місцезнаходжень в кеші (у відведеній для цього області). За алгоритмом LRU два елементи витісняються. Це вимагає одного біту для пари рядків кешу, де зазначається, який з них використовував останнім.

Кеш прямого відображення (Direct-mapped cache): застосовується для високошвидкісних кешів процесора, де не вистачає швидкодії 2-канального асоціативного кешування (рис. 9) [5]. Адреса нового елемента використовується тут для обчислення місцезнаходження в кеші (у відведеній для цього області, все, що було раніше, витісняється.

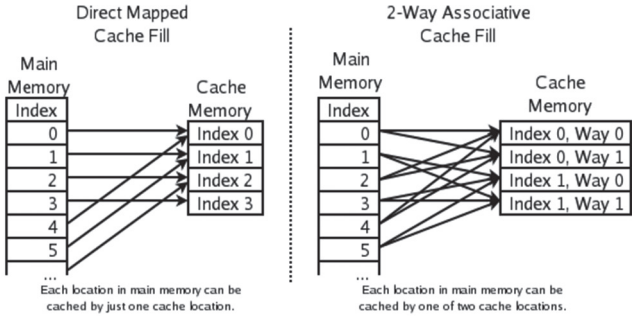

Рис. 9. Схеми роботи Direct-mapped та 2-Way Set Associative алгоритмів

Алгоритм многопоточного кешування даних (Multi Queue caching algorithm) додає ваги до елементів кешу (рис. 10). Зокрема, зважуються: елементи з різною вартістю зберігання, запит до яких і їх отримання вимагають багато часу; елементи, що мають різний розмір в кеші (кеш може спробувати витиснити більший елемент, щоб зберегти кілька менших елементів); елементи, що застарівають з плином часу (деякі кеші зберігають застарілу інформацію, наприклад, кеш новин, DNS-кеш або кеш веб-браузера). Комп'ютер може витиснити елементи внаслідок їх старіння. Залежно від розміру кешу, кешування нових елементів може підштовхнути витиснення старих [6].

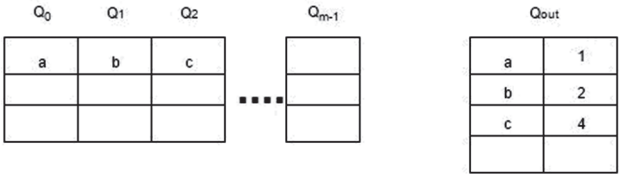

Рис. 10. Схема роботи MQ алгоритму

Існують також різні алгоритми для забезпечення когерентності кешу. Це застосовується тільки у випадках, коли декілька незалежних кешів використовується для зберігання однієї і тієї ж інформації (наприклад, декілька серверів баз даних оновлюють загальний файл даних).

4. Використання нейромереж для реалізації алгоритму Беладі

Для вирішення завдань кешування даних в інформаційних системах, що потребують використання значних потужностей, найбільш привабливим є застосування алгоритмів SLRU та MQCA, що в деякій мірі можна інтерпретувати як спрощені варіанти вирішення проблеми реалізації алгоритму Беладі. Слід відзначити, що їх безпосереднє використання за допомогою класичних

обчислювальних засобів потребує додаткових зусиль і не гарантує отримання оптимальних результатів. Доречним буде розглянути можливість нейромережевої реалізації алгоритму Беладі.

Як було відзначено вище, головною проблемою практичної реалізації цього алгоритму є у передбаченні часу використання кешу. Таке передбачення можна здійснити за допомогою нейронної мережі з додаванням необхідних часових параметрів процедури кешування. Розглянемо загальний сценарій використання таких параметрів для інформаційних систем підприємств та організацій, що, безумовно, залежить від конкретної предметної області.

Більшість підприємств та потужних організацій працюють зі своїми внутрішніми корпоративними системами протягом бізнес дня. Це означає, що потік робочих процесів та відповідних запитів до системи можна розділити на умовні ітерації. Тривалість такої ітерації буде залежати від початку та кінця роботи користувачів інформаційної системи в усіх часових поясах. Оскільки нейромережа, що передбачається використовувати для кешування за алгоритмом Беладі, має працювати з великим обсягом даних, то мінімальною одиницею виміру часу у відповідних процедурах передбачення доцільно вибрати годину.

Специфіка конкретних підприємств може передбачати повторення багатьох процесів обробки даних протягом року (наприклад, фінансові розрахунки є обов'язковими на початку та в кінці фінансового періоду). Це означає що в процедуру прогнозування слід додати параметр, що відповідає за місяць виконання запитів.

Інколи існує необхідність розглядати часові ітерації з урахуванням тижня виконання завдань обробки даних. Подібна організація роботи зустрічається не так часто, як помісячна, але також повинна братися до уваги. У такому разі додатковим параметром буде номер дня у тижні.

Треба відзначити, що організація бізнес процесів та ітерацій для інформаційних систем підприємств та організацій суттєво відрізняється. У деяких випадках існує чітке визначення поняття ітерації, що значно полегшує задачу визначення часу для збереження запиту. В реалізації такої системи поняття максимальної одиниці часу може становити день, а кількість можливих варіантів буде дорівнювати кількості днів у бізнес ітерації підприємства.

Для врахування людського фактору та можливих ризиків слід додати до параметрів прогнозуючої нейромережі спеціальний буфер, розмір якого цілком залежить від предметної області системи та значення якого може корегуватися під час роботи системи. Майже напевно цей параметр не буде перевищувати п'ятдесят відсотків від можливого діапазону значень часу і має бути більшим за один відсоток. Таким чином, доцільно використовувати часові параметри слід з буфером у двадцять п'ять відсотків від діапазону можливих значень часу. На рис. 11 наведено загальну схему роботи нейромережевого методу кешування з використанням алгоритму Беладі.

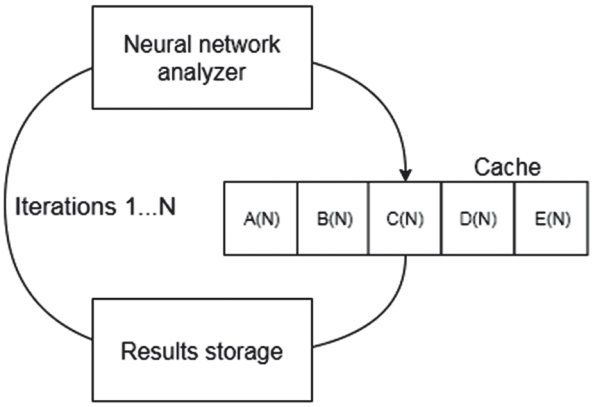

Рис. 11. Загальна схема роботи нейромережевого алгоритму кешування

Процедура Беладі реалізується з використанням нейромережевого аналізатора NNA (Neural network analyzer). Виходом NNA є послідовність номерів сторінок, що заміщаються в кеші A(N),..., E(N) при обробці запитів.

Від предметної області відповідної системи буде залежати, чи потрібно зберігати результат запиту після того, як він був виконаний користувачем один раз. У деяких випадках важливо тримати результат у пам'яті доки він вважається актуальним і готовим до використання. Можливою є ситуація, коли можна з упевненістю сказати, що результат запиту нас більше не зацікавить після того, як він був завантажений клієнтом вперше.

Таким чином, нейромережевий підхід може вирішити проблему алгоритму Беладі за умов визначення ефективних часових ваг та діапазонів, а також використання додаткових ресурсів.

5. Стислий аналіз роботи проекту-доказу концепції

Для перевірки доцільності використання нейронної мережі для передбачення запитів до інформаційної системи (згідно з алгоритмом Беладі) було розроблено проект-доказ концепції. З цією метою (за допомогою багатошарової персептронної нейромережі) було змодельовано аналітичну сервісну систему побудови складних транспортних маршрутів (АССТМ). Система дозволяє прокласти маршрут від однієї точки до іншої, враховуючи стан доріг, розклад громадського транспорту і багато інших параметрів. Користувачі сервісу – великі підприємства з тисячами вантажних перевозок на рік.

Очевидно, що сервіс не зможе миттєво відповідати на запити користувачів і буде використовувати

кеш, щоб прискорити хоча б наступні виклики запитів. У цьому разі нейромережа допомогає прискорити обробку першого виклику запиту конкретного маршруту клієнта.

Вхідні дані від користувача – перелік ключових точок маршруту (адреса або координати). Вихідні дані від сервісу – перелік усіх можливих маршрутів, які на даний момент використовують відомі перевізники по цим критеріям.

Навчання нейромережі та прогнозування відбувається після завершення бізнес дня у користувачів. Таким чином, навантаження на систему є рівномірним протягом доби і не заважає користувачам. Прогноз розраховується на наступний бізнес день.

Підсистема АССТМ, що визначає маршрути, спочатку знаходить найближчу відому їй точку у маршрутній базі та прокладає шлях від неї до наступної відомої точки. Відрізки від відомих точок до введених користувачем будуються окремо, використовуючи менший обсяг необхідної інформації. Тож маємо фіксований набір вхідних параметрів – перелік відомих системі точок у запиті, що відповідають початку шляху, кінцю і необхідним зупинкам. Для пілотного проекту на відображення цих точок достатньо буде виділити вектор на сто тисяч елементів. Другий важливий компонент вхідних даних – стан ринку перевозок. З урахуванням рекомендацій теорії логістики, далі будується продовження вектору на тисячу елементів. До вихідних даних нейромережі додано час виклику запитів.

Вибірку для навчання було згенеровано випадково і скореговано так, щоб можна було простежити умовну закономірність навантаження на систему.

Готову реалізацію нейромережі було взято з бази Encog Machine Learning Framework, що має можливість інтеграції з середовищем Java та містить інформаційні приклади. Тестові дані були збережені у PostgreSQL базі даних. Так як мета проекту полягає, насамперед, у перевірці концепції, нейромережа має лише один прихований шар.

Після запуску системи нейромережі доводилося приймати рішення, чи буде необхідним результат певного запиту наступного бізнес дня. Для перевірки використовувалися змінені дані з навчальної вибірки. Для навчання та роботи нейромережі з проекту-доказу концепції знадобилося близько восьми гігабайтів.

Кінцева версія налаштованої нейромережі змогла заздалегідь підготувати п'ятдесят сім зі ста запитів, а ще двадцять були підготовані помилково. Враховуючи, що дані були згенеровані випадково і не відповідають реальній ситуації, а також те, що для навчання використовувалися ресурси набагато менші за потенційні ресурси кінцевих користувачів, такий результат можна назвати успішним. Звичайно, обрана предметна область набагато ширше за спрощену модель даних у проекті-доказу концепції, але експеримент все-одно можна вважати вдалим. За наявністю достатніх ресурсів та спеціалістів у галузі конкретного питання можна побудувати нейромережу, що буде значно ефективніше реалізовувати процедуру кешування даних за алгоритмом Беладі.

Висновки

У статті був розглянутий новий спосіб кешування запитів за допомогою нейронної мережі. Цей спосіб дозволяє вирішити проблему алгоритму Беладі і досить влучно прогнозувати час актуальності збереження запиту. Запропонований алгоритм потребує дещо більше ресурсів, ніж існуючі аналоги, але дозволяє економити час клієнта, що користується аналітичною системою. Результати пілотного проектування свідчать про те, що застосовування нейромережевого типу кешування має сенс тільки для великих систем, що потребують багато часу на підготовку відповіді. Також важливим фактором доцільності такого застосування є наявність достатніх обчислювальних ресурсів. Для реалізації нейромережевого типу кешування необхідним є попередній аналіз предметної області. Слід також відзначити, що обсяг необхідної обчислювальної потужності збільшується не прямо пропорційно обсягу даних в інформаційній системі, тож реалізація такого підходу буде особливо вигідна для корпоративних систем.

Список літератури:

1. *Чумаченко П.В.* Мережі доставки контенту // П.В. Чумаченко, Т.А. Ліхоузова, О.І. Лісовиченко / Міжвідомчий науково-технічний збірник «Адаптивні системи автоматичного управління» № 2(29), 2016 – с. 78-89. **2.** *J. Z. Teng and R. A. Gumaer,* «Managing IBM database 2 buffers to maximize performance», IBM Sys. J., vol. 23, no. 2, pp. 211–218, 1984. **3.** *P. Cao and S. Irani,* «Cost-aware WWW proxy caching algorithms», in Proc. USENIX Symp. Internet Technologies and Systems, Monterey, CA, 1997. **4.** *L. Degenaro, A. Iyengar, I. Lipkind, and I. Rouvellou,* «A middleware system which intelligently caches query results», in Middleware 2000, vol. LNCS 1795, pp. 24–44, 2000. **5.** *J. D. Gee, M. D. Hill, D. N. Pnevmatikatos, and A. J. Smith,* «Cache performance of the SPEC benchmark suite», Tech. Rep. CS-TR-1991-1049, University of California, Berkeley, 1991. **6.** *M. N. Nelson, B. B. Welch, and J. K. Ousterhout,* «Caching in the Sprite network file system», ACM Transactions on Computer Systems, vol. 6, no. 1, pp. 134–154, 1988.

Надійшла до редколегії 16.03.2018