УДК 009.4

Б.М. Гавриш [1], к.т.н., доцент
Б.В. Дурняк [1], д.т.н., професор
О.В. Тимченко [1, 2], д.т.н., професор

ПОБУДОВА ІЄРАРХІЧНИХ СЦЕНАРІЇВ ОПРАЦЮВАННЯ ДАНИХ

Abstract. Data processing techniques are considered to reduce analysis time and reduce memory requirements by splitting data into segments and constructing a hierarchical model scenario for each segment.

Keywords: data segmentation, hierarchical algorithms

Вступ

Зазвичай, коли йдеться про серйозне аналітичне опрацювання, особливо якщо використовується термін Data Mining, мається на увазі, що даних величезна кількість. Не існує універсальних методів аналізу або алгоритмів, придатних для будь-яких випадків і будь-яких обсягів інформації. Методи аналізу даних істотно відрізняються один від одного за продуктивністю, якістю результатів, зручністю застосування і вимогами до даних. Оптимізація може проводитися на різних рівнях: обладнання, бази даних, аналітична платформа, підготовка вихідних даних, спеціалізовані алгоритми. Аналіз великого обсягу даних вимагає особливого підходу, оскільки технічно складно їх переробити за допомогою тільки з використанням більш потужного устаткування.

Звичайно, можна збільшити швидкість опрацювання даних за рахунок більш продуктивного обладнання, тим більше, що сучасні сервера і робочі станції використовують багатоядерні процесори, оперативну пам'ять значних розмірів і потужні дискові масиви. Однак, є безліч інших способів опрацювання великих обсягів даних, які дозволяють підвищити масштабованість і не вимагають нескінченного оновлення обладнання.

Метою роботи є розгляд і побудова ієрархічного сценарію опрацювання даних для зменшення часу аналізу і зниження вимог до пам'яті, зокрема при стисненні великих обсягів даних.

Основна частина

Сучасні бази даних включають різні механізми, застосування яких дозволить значно збільшити швидкість аналітичного опрацювання:

- Попереднє обчислення даних. Відомості, які найчастіше використовуються для аналізу, можна заздалегідь обчислити і в

[1], Українська академія друкарства
[2] Uniwersytet Warmińsko-Mazurski w Olsztynie

підготовленому для опрацювання вигляді зберігати на сервері БД у вигляді багатовимірних кубів, матеріалізованих представлень, спеціальних таблиць.

- Кешування таблиць в оперативну пам'ять. Дані, які займають небагато місця, але до яких часто звертаються в процесі аналізу, наприклад, довідники, можна засобами бази даних кешувати в оперативну пам'ять. Так у багато разів скорочуються звернення до більш повільної дискової підсистеми.
- Розбиття таблиць на розділи і табличні простори. Можна розміщувати на окремих дисках дані, індекси, допоміжні таблиці. Це дозволить СУБД паралельно зчитувати і записувати інформацію на диски. Крім того, таблиці можуть бути розбиті на розділи (partition) таким чином, щоб при зверненні до даних була мінімальна кількість операцій з дисками. Наприклад, якщо найчастіше ми аналізуємо дані за останній місяць, то можна логічно використовувати одну таблицю з історичними даними, але фізично розбити її на кілька розділів, щоб при зверненні до місячних даних зчитувався невеликий розділ і не було звернень до всіх історичних даних.

Це тільки частина можливостей, які надають сучасні СУБД. Підвищити швидкість вилучення інформації з бази даних можна і десятком інших способів: раціональне індексування, побудова планів запитів, паралельна обробка SQL запитів, застосування кластерів, підготовка аналізованих даних за допомогою збережених процедур і тригерів на стороні сервера БД тощо.

Можливості підвищення швидкості не зводяться лише до оптимізації роботи бази даних, багато чого можна зробити за допомогою комбінування різних моделей. Відомо, що швидкість опрацювання істотно пов'язана із складністю використовуваного математичного апарату. Чим простіші механізми аналізу використовуються, тим швидше аналізуються дані.

Можлива побудова сценарію опрацювання даних таким чином, щоб дані "проганялись" через сито моделей. Тут застосовується проста ідея: не витрачати час на опрацювання того, що можна не аналізувати.

Спочатку використовуються найбільш прості алгоритми. Частина даних, які можна опрацювати за допомогою таких алгоритмів і які не потрібно опрацьовувати з використанням більш складних методів, аналізується і виключається з подальшого опрацювання. Решта даних передаються на наступний етап опрацювання, де використовуються більш складні алгоритми, і так далі по ланцюжку. На останньому вузлі сценарію опрацювання застосовуються найскладніші алгоритми, але обсяг аналізованих даних у багато разів менший за початкову вибірку. В результаті загальний час, необхідний для опрацювання всіх даних, зменшується на порядки.

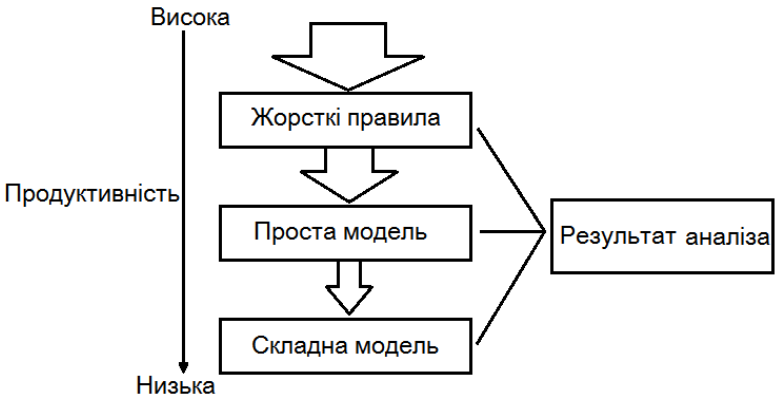

Рис. 1. Комбінування моделей, прогон даних через сито моделей

Також ефективною стратегією опрацювання великих обсягів даних є розбиття даних на сегменти і побудова моделей для кожного сегмента окремо, з подальшим об'єднанням результатів. Найчастіше у великих обсягах даних можна виділити кілька підмножин, які відрізняються між собою.

У цьому випадку замість побудови однієї складної моделі для всіх можна будувати кілька простих для кожного сегмента. Подібний підхід дозволяє підвищити швидкість аналізу і знизити вимоги до пам'яті завдяки опрацюванню менших обсягів даних в один прохід. Крім того, в цьому випадку аналітичну обробку можна розпаралелити, що теж позитивно позначається на витрачений час. До того ж моделі для кожного сегмента можуть будувати різні аналітики.

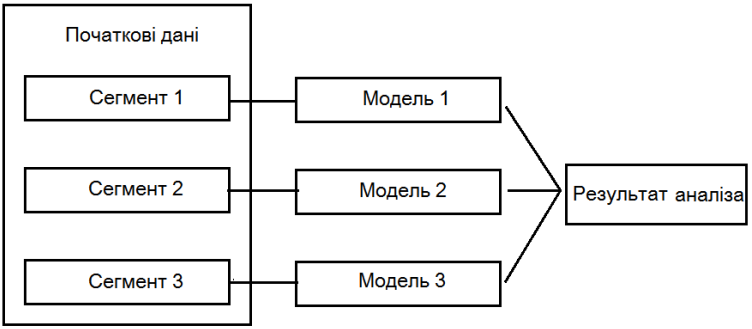

Рис. 2. Розбиття даних на сегменти і побудова моделей для кожного сегмента окремо, з подальшим об'єднанням результатів

При передачі та збереженні великих обсягів інформації надмірність відіграє негативну роль, оскільки вона не тільки призводить до збільшення

часу передачі і функціональної надійності передачі інформації та її зберігання, а й до зростання сукупної вартості. В зв'язку з цим на сьогоднішній день для забезпечення ефективності передачі великих обсягів інформації та зберігання широко використовуються різноманітні способи стиснення.

Але також при стисненні даних виникає ситуація, коли частина даних втрачається. Інколи ці втрати не відіграють значної ролі, але бувають ситуації, коли втрата даних може стати критичною. Саме тому способи стиснення без втрат користуються популярністю та постійно розвиваються.

Особливо такі способи важливі під час стиснення великих обсягів даних, коли постає необхідність зменшити розмір оригінальних даних, але при цьому мати можливість відновити стиснені дані без втрати.

Стиснення базується на усуненні надлишку інформації, яка міститься у вихідних даних. Наприклад, повторення в тексті фрагментів (наприклад, слів природної або машинної мови). Подібний надлишок зазвичай усувається заміною повторюваних послідовностей коротшим значенням (кодом). Інший вид надлишковості пов'язаний з тим, що деякі значення в даних, які ущільняються, трапляються частіше інших, при цьому можна замінювати дані, що часто трапляються, коротшими кодами, а ті, що рідше – довшими (ймовірнісне стиснення). Стиснення даних, які не мають властивості надлишку (наприклад випадковий сигнал чи шум), неможливе. Також, зазвичай, неможливо стиснути зашифровану інформацію [1].

Різні алгоритми стиснення можуть вимагати різної кількості ресурсів обчислювальної системи, на яких їх застосовують [2]:

1) оперативної пам'яті під проміжні дані;

2) постійної пам'яті під код програми і константи;

3) процесорного часу.

У цілому ці вимоги залежать від складності та «інтелектуальності» алгоритму. Загальна тенденція така: чим більш ефективний та універсальний алгоритм, тим більші вимоги до обчислювальних ресурсів він висуває. Тим не менш, у специфічних випадках прості і компактні алгоритми можуть працювати не гірше складних і універсальних. Системні вимоги визначають їх споживчі якості: чим менш вимогливий алгоритм, тим у простішій, а отже, компактній, надійній і дешевій системі він може працювати.

Оскільки алгоритми стиснення і відновлення працюють в парі, має значення співвідношення системних вимог до них. Можна, ускладнивши один алгоритм, значно спростити інший. Таким чином, існують три наступні варіанти [2].

1. Алгоритм стиснення вимагає більших обчислювальних ресурсів, ніж алгоритм відновлення. Таке співвідношення більш поширене та характерне для випадків, коли одноразово стиснені дані будуть використовуватися багато разів. Як приклад можна навести цифрові аудіо- та відеопрогравачі.

2. Алгоритми стиснення і відновлення вимагають приблизно рівних обчислювальних ресурсів. Найбільш прийнятний варіант для ліній зв'язку,

коли стиснення і відновлення відбувається одноразово на двох її кінцях (наприклад, в цифровій телефонії).

3. Алгоритм стиснення істотно менш вимогливий, ніж алгоритм відновлення. Така ситуація характерна для випадків, коли процедура стиснення реалізується простим, часто портативним пристроєм, для якого обсяг доступних ресурсів дуже критичний, наприклад, космічний апарат або велика розподілена мережа датчиків. Це можуть бути також дані, розпакування яких потрібно в дуже малому відсотку випадків, наприклад запис камер відеоспостереження.

Може виникнути ситуація, коли необхідно стиснути дані невідомого формату [2].

Існують два основних підходи для стиснення даних невідомого формату.

1. На кожному кроці алгоритму стиснення черговий стискуваний символ або поміщається у вихідний буфер стискального кодера як є (зі спеціальним прапором для позначення, що він не був стиснутий), або група з кількох стискуваних символів замінюється посиланням на відповідну групу з уже закодованих символів. Оскільки відновлення стиснутих таким чином даних виконується дуже швидко, такий підхід часто використовується для створення саморозпаковувальних програм.

2. Для кожної послідовності символів, яку необхідно стиснути, одноразово або в кожний момент часу збирається статистика її появи в кодованих даних. На основі цієї статистики обчислюється ймовірність значення чергового кодованого символу (або послідовності символів). Після цього застосовується той чи інший різновид ентропійного кодування, наприклад, арифметичне кодування або кодування Хаффмана, для представлення частіших послідовностей короткими кодовими словами, а рідших — довшими.

Висновки

Наведені алгоритми використовують для одновимірних даних. Вони не підходять для стиснення тривимірних (3-D) даних. Оскільки для таких даних (особливо для медичних 3-D зображень) важливо, щоб відновлені після стиснення дані були ідентичні оригінальним, то для їх стиснення необхідно використовувати алгоритми без втрат..

1 *К. Шеннон.* Теория связи в секретных системах // Работы по теории информации и кибернетике – М.: ИЛ, 1963. — С. 243-322. — 830 с.

2 *Шнайер Б.* Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си – М.: Триумф, 2002. — 816 с.

http://doi.org/10.5281/zenodo.3610665

Поступила 1.08.2019р.