Реализация алгоритма хеширования с учетом местоположения средствами Pl/PgSQL

Н.П. Плотникова, В.А. Кевбрин

Мордовский государственный университет имени Н.П. Огарёва

Аннотация: Нашу жизнь пронизывают данные, бесконечные потоки информации проходят через компьютерные системы. Сегодня нельзя представить современное программное обеспечение без взаимодействия с базами данных. Существует много различных систем управления базами данных в зависимости от цели использования информации. В статье рассматривается алгоритм хеширования с учетом местоположения на основе языка Pl/PgSQL, который позволяет искать похожие документы в базе.
Ключевые слова: хэширование, поле, строка, текстовые данные, запрос, программное обеспечение, язык структурированных запросов.

Введение

Вопрос хранения большого количества документов и эффективного поиска по ним, несмотря на быстрые темпы развития информационных инфраструктур, всё ещё является серьезной проблемой. Существует несколько подходов к решению данной задачи. Один из вариантов – хранение текстовых данных в качестве отдельных полей напрямую в базе данных (БД). Это позволяет конструировать запросы, которые будут сканировать или каким-либо образом использовать их напрямую. Этот подход имеет ряд серьезных недостатков, таких как:

1) Организация хранения документов в системах управления базами данных (СУБД) [1]. Многие системы управления базами данных не предполагают сохранения больших объемов данных в одном поле и им приходится реализовывать различного рода механизмы, которые будут позволять обрабатывать подобные данные. Например, PostgreSQL [2] использует механизм toast-таблиц.
2) Следствие первого пункта данного списка – очень сложный, иногда, неэффективный поиск, особенно по частичному совпадению.

Алгоритм

Алгоритм хеширования с учетом местоположения (locality-sensitive hashing - LSH) создан для поиска документов по неполному соответствию, рассмотрим все его этапы подробно:

1) Шинглирование данных [3] – разбиение информации определенным образом, чтобы получился массив текстовых данных. Чаще всего разбивают одним из двух способов:
    a. По количеству символов.
    b. По токенам.
2) Преобразование массива текстовых данных в массив чисел. Например, при помощи хэширования [4] или простого сопоставления по словарю.
3) Min hashing [5] – создается массив перестановок встреченных данных и используется one hot encoding [6].
4) Получившийся массив дробится в зависимости от конфигурации алгоритма по частям и проводится анализ схожести либо хэшей, сгенерированных на основе частей, либо самих частей между различными документами. В зависимости от количества найденных совпадений, выносится вердикт о схожести документов.

Реализация на Pl/PgSQL

В качестве СУБД использовалась PostgreSQL, которая позволяет писать расширения как на языке C, так и при помощи функций на языке Pl/PgSQL[7]. Структурно необходимы следующие таблицы:

1) Dictionary – для хранения соответствия токенов числовому значению.
2) Lsh_config – для данных о конфигурации алгоритма:
    a. Итоговое количество частей.

b. Минимальное количество совпавших частей, которое определяет, похожи ли документы.
c. Количество документов, выдаваемых при поиске.
d. Количество перестановок для min hashing.
e. Перестановки.

3) Lsh_parts – результаты обработки.
4) Lsh_tables – информация о таблицах и полях, на основе которых рассчитан алгоритм.

Для полноценной реализации потребовалось написание нескольких функций, которые соответствуют описанию этапов LSH:

1) Tokenization – токенизация данных, написать эту функцию средствами языка структурированных запросов (SQL) тяжело, поэтому использовалось расширение plpython3u с пакетами nltk и pymorphy3.
2) Update_dict – для обновление словаря.
3) Generate_min_hash_funcs – генератор перестановок.
4) One_hot – преобразование данных в соответствии с алгоритмом первого вхождения элемента.
5) Min_hash – реализация min hashing.
6) Набор вспомогательных функций для работы с алгоритмом и структурой БД:
   a. Lsh – инициализация структуры БД;
   b. Partition_lsh – добавление партиционирования таблицы lsh_parts для наиболее эффективного поиска.
   c. Reindex_lsh – пересчет всех данных в соответствии с измененной конфигурацией или словарем.
   d. Lsh_find – поиск документов, показан на рис.1.

```
CREATE OR REPLACE FUNCTION lsh_find(doc_in text, table_name_in text, column_name_in text) RETURNS jsonb[] AS
$$

DECLARE
    _count_elems    int;
    _result         jsonb[];
    _min_hash       int[];
    _id             int;
    _find           jsonb = jsonb_build_array();
    _min_find       int;
    _limit_find     int;
    _count_parts    int;
BEGIN
    SELECT min_hash_func_count/count_parts, count_parts, min_find, limit_find
    FROM lsh_config
    INTO _count_elems, _count_parts, _min_find, _limit_find;

    _min_hash = min_hash( tokens: one_hot( tokens: tokenization_py( doc: doc_in)));
    FOR i IN 1..(SELECT count_parts FROM lsh_config) LOOP
        SELECT _find || jsonb_agg(jsonb_build_object('doc_id', doc_id, 'part_id', part_id))
        FROM lsh_parts
        WHERE part_id = i
          AND min_hash = (_min_hash[(i-1)*_count_elems:i*_count_elems])::text
        INTO _find;
    END LOOP;

    RAISE NOTICE '_find %', _find;

    EXECUTE '
    WITH tmp as (
        SELECT (value->>''doc_id'')::bigint as doc_id, count((value->>''part_id'')::int) as count
        FROM jsonb_array_elements($1)
        GROUP BY (value->>''doc_id'')::bigint
        HAVING count((value->>''part_id'')::int) > $2
        ORDER BY count((value->>''part_id'')::int) desc
        LIMIT $3
    )
    SELECT array_agg(jsonb_build_object(''doc'', tni.'||column_name_in||', ''probability'', tmp.count::numeric/$4))
    FROM tmp, '||table_name_in||' tni
    WHERE tmp.doc_id = tni.id'
    USING _find, _min_find, _limit_find, _count_parts
    INTO _result;

    RETURN _result;
END;
$$ LANGUAGE plpgsql;
```

Рис. 1. – Листинг кода поиска документов

Как видно из приведённого выше листинга, в реализации широко используется механизм партиционирования PostgreSQL. Дробление данных происходит в зависимости от номера части и имени таблицы, в которой находятся исходные данные.

Данный вариант реализации накладывает некоторые ограничения, например, в исходных данных обязано присутствовать поле id. В современных БД оно присутствует почти во всех таблицах, потому является очень мягким недостатком.

Основная проблема алгоритма состоит в постоянной необходимости пересчитывать все данные при добавлении нового элемента в словарь. Это делает алгоритм очень тяжелым в использовании на большом объеме данных. С другой стороны, при накоплении большого словаря, с каждым новым документом, вероятность появления нового токена сильно снижается, что делает его очень удобным при работе с однотипными данными, например, универсальный передаточный документ (УПД)[8].

Тестирование

Для тестирования работоспособности были взяты текстовые данные в формате json [9, 10] в количестве 10000 строк и общим объемом 500 Мб. Пример данных приведен на рис.2.

```
16805662 {"aux": null, "doc_id": 16805661, "job_id": 5654219, "msg_id": "d2df0cc6-2e4'
16805661 {"body": {"task_id": "REF.1879", "client_list": [{"bik": "", "dsc": "", "inn'
16805660 [{"item_list": [{"item_list": [{"cnt": "1041", "batch": "Y170224", "nom_typ"
16805659 {"msg_id": "70139efb-fad0-45d4-89ce-f9c5d2059ef6", "msg_type": "REFERENCE",
16805657 [{"item_list": [{"item_list": [{"cnt": "1041", "batch": "Y170224", "nom_typ"
16805656 [12971057, 12973117, 12973118, 12973119, 12973120, 12973121, 12973122, 12973
16805655 [13008875, 13009041, 12973133, 12973132, 13008859, 12973122, 13097078, 12973
```

Рис. 2. –Пример исходных данных

Описание тестового стенда:

1) WSL 2.
2) Docker [11].
3) CPU – AMD 3900X.
4) RAM – 64 Gb.
5) SSD NVMe.

В таблице 1 представлены результаты проведенного тестирования.

Таблица № 1

Результаты тестирования LSH на базе Pl/PgSQL

Процесс	Стандартная таблица, с	Стандартная таблица с индексацией, с	LSH, с
Обработка 10000 записей	35	37	7686
Добавление 1 элемента к уже существующим данным	0,023	0,026	7635
Поиск документа	2,914	2,863	0,916

По полученным результатам можно сделать следующие выводы:

1) Алгоритм обработки выполняется слишком долго и требует серьезной оптимизации.
2) Поиск подходящих документов очень эффективен.

Заключение

По итогу многочисленных экспериментов был получен результат работы со временем поиска 0,916 секунды, что очень эффективно. Стоит отметить следующие особенности:

1) Длительное время обработки связано с особенностями Pl/PgSQL:
    a. Стоит избегать использования циклов.
    b. Максимальный размер массива жестко ограничен, а алгоритму с постоянно пополняющимся словарем этого мало.

c. Вызов plpython отнимает время.

2) Из пункта 1 вытекает следующее – выполнять реиндексацию крайне невыгодно. Следует это делать только по накоплению определенного количества новых элементов словаря, в таком случае добавление одного нового документа не потребует полной реиндексации, и время уменьшится до 2,955 с.

Таким образом, на данном этапе проработки, алгоритм больше подходит для эффективного поиска документов в заранее обработанном архиве данных, нежели для работы на постоянно пополняющихся БД.

Литература

1. Плотникова Н.П., Мартынов В.А. Применение дерева отрезков в PostgreSQL // Инженерный вестник Дона. 2023. №9. URL: ivdon.ru/ru/magazine/archive/n9y2023/8684

2. Hellerstein J.H. Looking back at Postgres // Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker. 2018. P. 205–224.

3. Игнатов Д.И., Кузнецов С.О. Разработка и апробация системы поиска дубликатов в текстах проектной документации // Бизнес-информатика. 2008. №4. URL: cyberleninka.ru/article/n/razrabotka-i-aprobatsiya-sistemy-poiska-dublikatov-v-tekstah-proektnoy-dokumentatsii

4. Плотникова Н.П., Кевбрин В.А., Болохов Д.А. Дедупликация больших объемов данных при помощи баз данных // Инженерный вестник Дона. 2023. №9. URL: ivdon.ru/ru/magazine/archive/n9y2023/8676

5. Авезова Я.Э. Современные подходы к построению хеш-функций на примере финалистов конкурса SHA-3 // НиКа. 2015. №3 (11). URL: cyberleninka.ru/article/n/sovremennye-podhody-k-postroeniyu-hesh-funktsiy-na-primere-finalistov-konkursa-sha-3-1

6. Firsanova V.I. Automatic recognition of messages from virtual communities of drug addicts // Journal of applied linguistics and lexicography. 2020. №1. URL:

cyberleninka.ru/article/n/automatic-recognition-of-messages-from-virtual-communities-of-drug-addicts (дата обращения: 20.06.2024).

7. Джиджоев В.М., Бучацкий Р.А., Пантилимонов М.В., Томилин А.Н. Динамическая компиляция пользовательских функций на языке pl/pgsql // Труды ИСП РАН. 2020. №5. URL: cyberleninka.ru/article/n/dinamicheskaya-kompilyatsiya-polzovatelskih-funktsiy-na-yazyke-pl-pgsql

8. Сопнева Я.С. Универсальный передаточный документ(упд) и его неотъемлемость при решении налоговых споров // Инновационная наука. 2020. №2. URL: cyberleninka.ru/article/n/universalnyy-peredatochnyy-dokument-upd-i-ego-neotemlemost-pri-reshenii-nalogovyh-sporov

9. Crockford D. JSON: The Fat-Free Alternative to XML // Boston 2006. URL: json.org/fatfree.html#:~:text=JSON%3A%20The%20Fat%2DFree%20Alternative%20to%20XML

10. Наумов Р.К., Железков Н.Э. Сравнительный анализ форматов хранения текстовых данных для дальнейшей обработки методами машинного обучения // Научный результат. Информационные технологии. 2021. №1. С. 40-47.

11. Karim Z. Take the confusion out of Docker, VMs, and microservices // IBM Developer 2019 URL: developer.ibm.com/articles/breaking-down-docker-and-microservices

References

1. Plotnikova N.P., Martynov V.A. Inzhenernyj vestnik Dona. 2023. №9. URL: ivdon.ru/ru/magazine/archive/n9y2023/8684

2. Hellerstein J.H. Looking back at Postgres. Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker. 2018. P. 205–224.

3. Ignatov D.I., Kuznetsov S.O. Biznes-informatika. 2008. №4. URL: cyberleninka.ru/article/n/razrabotka-i-aprobatsiya-sistemy-poiska-dublikatov-v-tekstah-proektnoy-dokumentatsii

4. Plotnikova N.P., Kevbrin V.A., Bolohov D.A. Inzhenernyj vestnik Dona. 2023. №9. URL: ivdon.ru/ru/magazine/archive/n9y2023/8676.

5. Avezova Ja.Je. NiKa. 2015. №3 (11). URL: cyberleninka.ru/article/n/sovremennye-podhody-k-postroeniyu-hesh-funktsiy-na-primere-finalistov-konkursa-sha-3-1.

6. Firsanova V.I. Journal of applied linguistics and lexicography. 2020. №1. URL: cyberleninka.ru/article/n/automatic-recognition-of-messages-from-virtual-communities-of-drug-addicts

7. Dzhidzhoev V.M., Buchackij R.A., Pantilimonov M.V., Tomilin A.N. Trudy ISP RAN. 2020. №5. URL: cyberleninka.ru/article/n/dinamicheskaya-kompilyatsiya-polzovatelskih-funktsiy-na-yazyke-pl-pgsql

8. Sopneva Ja.S. Innovacionnaja nauka. 2020. №2. URL: cyberleninka.ru/article/n/universalnyy-peredatochnyy-dokument-upd-i-ego-neotemlemost-pri-reshenii-nalogovyh-sporov

9. Crockford D. JSON: The Fat-Free Alternative to XML. Boston: 2006. URL: json.org/fatfree.html#:~:text=JSON%3A%20The%20Fat%2DFree%20Alternative%20to%20XML

10. Naumov R.K., Zhelezkov N.E. Nauchnyj rezul'tat. Informatsionnye tekhnologii. 2021. №1. pp. 40-47.

11. Karim Z. Take the confsion out of Docker, VMs, and microservices. IBM Developer 2019. URL: developer.ibm.com/articles/breaking-down-docker-and-microservices.

Дата поступления: 25.05.2024

Дата публикации: 11.07.2024