Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
«Высшая школа экономики»

Факультет компьютерных наук
Основная образовательная программа
Прикладная математика и информатика

КУРСОВАЯ РАБОТА

Программный проект на тему

"Структуры данных для вероятностной фильтрации по подзапросам в ClickHouse"

Выполнил студент группы 176, 3 курса,
Ибрагимов Рузель Ильфакович

Руководитель КР:
Кандидат технических наук, доцент
Сухорослов Олег Викторович

Консультант КР:
Руководитель группы разработки ClickHouse
Миловидов Алексей Николаевич

Москва 2020

Abstract

In the current world, it is increasingly common for analysts to create complex queries to databases that have changed significantly since their creation. Sometimes such requests can be very large and take a very long time to complete. Therefore, database developers try to expand their functionality so that clients who write queries to the database can speed up their queries themselves. The introduction of one of these features, namely the use of probability filters, is described in this article.
Keywords: databases, filtering queries, probabilistic filtering, Bloom filter, Cuckoo filter, Vacuum filter.

Аннотация

В текущем мире все чаще аналитики создают сложные запросы в базы данных, которые сильно изменились с момента их создания. Иногда такие запросы могут быть очень большими и выполняться очень долго. Поэтому разработчики баз данных стараются расширить свою функциональность, чтобы клиенты, которые пишут запросы в базу данных, имели возможность самим ускорять свои запросы. О внедрении одной из таких функциональностей, а именно о применении вероятностных фильтров, и рассказывается в этой статье.
Ключевые слова: базы данных, запросы фильтрации, вероятностная фильтрация, фильтры Блума, фильтр кукушки, Vacuum filter.

Содержание

1 Введение

1.1 Описание предметной области

СУБД ClickHouse [1] отличается от привычных баз данных. Например, в широко известном PostgreSQL [2] данные хранятся построчно. Одна строка является непрерывной последовательностью байт, хранящихся на жестком диске компьютера. И чтение такой таблицы может быть выполнено только построчно. Такое размещение данных не является эффективным способом, если пользователю нужно часто выполнять агрегирующие функции. Например, вычислить среднее, медиану, среднее квадратическое отклонение, максимум, минимум, N-ю порядковую статистику и т.д. Такие задачи часто возникают у аналитиков. Чтобы выполнить агрегирующую функцию по одной колонке, база данных на основе PostgreSQL должна прочитать полностью всю таблицу. Это неэффективно. Гораздо быстрее будет, если СУБД будет считывать только необходимую колонку. Отталкиваясь от причин, написанных выше, разработчики решили создать колоночную базу данных – ClickHouse. Колоночная база данных – это не инновация ClickHouse, такие базы данных создавались и раньше, например Vertica [3], ParAccel [4] и другие. Колоночная база данных хранит каждую колонку последовательностью байт прямо в памяти, не обязательно вся колонка хранится целиком в одном месте, важно только, чтобы эту колонку можно было прочитать в виде последовательности.

1.2 Постановка задачи

Необходимо добавить в ClickHouse возможность вероятностной фильтрации. Фильтрация устроена следующим образом. На вход даются два множества, например, A и B. Следует построить такое множество , в котором лежат только элементы множества A, которые есть в множестве B. Менее формальное описание задачи: даны два множества, скажите для каждого элемента первого множества лежит ли он во втором множестве. Вероятностная филь-

трация – это вероятностное решение задачи, сформулированной выше. Для каждого элемента множества A нужно сказать, лежит ли он в множестве B, но при этом можно ошибаться с вероятностью p.

Необходимо предложить несколько вариантов решения данной задачи, каждый из которых должен быть оптимальным либо по своей скорости работы, либо по количеству потребляемой памяти при равных значениях p. Для гарантии оптимальности необходимо написать тесты.

С технической стороны стоит реализовать вычисление хэш-функции сложного объекта, который может являться композитом из разных типов колонок.

1.3 Актуальность и значимость работы

Ежедневно большое число аналитиков используют ClickHouse для своих подсчетов. Запросы могут быть самыми разными, но рассмотрим конкретно фильтрующие запросы. Например, клиент хочет посчитать функцию по элементам, которые лежат в колонке A и которые есть в колонке B. Чтобы решить такую задачу, базе данных сначала нужно определить множество элементов колонки A, которые есть в колонке B. Предположим, что задача поменялась и клиенту важно не точное значение множества $A \cup B$, а приблизительное. Именно в этом случае полезна вероятностная фильтрация – запрос вероятностной фильтрации выполнится значительно быстрее, чем обычная фильтрация (то есть получение точного значение множества $A \cup B$). В ClickHouse на данный момент отсутствует возможность выполнить вероятностную фильтрацию.

Приведем пример конкретной задачи. Например, в таблице хранятся запросы пользователей, а именно хранятся IP-адреса, текст запроса, время, в который был отправлен запрос. В другой таблице хранятся IP-адреса зарегистрированных пользователей. Пользователь ClickHouse хочет узнать, сколько пользователей из первой таблицы зарегистрированы в системе. Он делает запрос, который на языке SQL выглядит как-то так:

```
SELECT ip_adr FROM tableA WHERE ip_adr IN tableB;
```

Но задача может быть неточной, например, пользователю необходимо лишь оценить, что доля зарегистрированных пользователей превышает 50% от всех запросов. В таком случае можно было бы задать вероятностный запрос, который выглядел бы так:

```
SELECT ip_adr FROM tableA
WHERE ip_adr BLOOMFILTER_IN(2000) tableB;
```

`BLOOMFILTER_IN(2000)`– в данном случае фильтр Блума, состоящий из 2000 бит.

Второй запрос выполнился бы быстрее, чем первый, то есть клиент получил бы ответ на свой вопрос потратив меньше времени.

2 Обзор существующих решений

2.1 Фильтр Блума

Фильтр Блума [5] – это вероятностная структура данных, позволяющая проверять элемент на принадлежность множеству. Такая структура может вернуть ложноположительный ответ – это значит, что элемента в множестве нет, но фильтр Блума может ответить, что элемент в множестве есть. При этом поддерживаются только операции добавления элемента в множество и проверка принадлежности, таким образом удалить элемент из множества нельзя. Для реализации фильтра Блума необходим массив из последовательных битов. В реальности многоразовое обращение к большому массиву данных, которые лежат в памяти последовательно, выполняется очень быстро, поэтому фильтр Блума очень хорошо подходит для решения исходной задачи. Фильтр Блума представляется как массив из m бит. Также пользователь этой структуры должен определить k хэш-функций $h_i(x)$, каждый из которых будет отображать элемент в какой-то из m битов. При добавлении элемента x в множество необходимо выставить в 1 все биты, которые лежат на позициях

$h_1(x) \dots h_k(x)$. Для проверки принадлежности элемента множеству необходимо проверить, что все биты на позициях $h_1(x) \dots h_k(x)$ равны 1. Если это не так, то элемент точно не принадлежит множеству, если же все биты на этих позициях оказались в положении 1, то элемент может принадлежать множеству.

Рассмотрим вероятность ложноположительного срабатывания. Пусть размер битового массива равен m, а количество независимых попарно хэш-функций $h_1(x) \dots h_k(x)$ равно k.

Каждая функция равномерно определяет бит p для элемента x, то есть $\forall i \; P(h_i(x) = p) = \frac{1}{m}$.

Вероятность того, что в бит p не будет записана единица при очередной операции вставки равна $P(h_1(x) \neq p, \dots h_k(x) \neq p) = P(h_1(x) \neq p)^k = (1 - \frac{1}{m})^k$.

Вероятность того, что в бит p не будет записана единица после вставки n элементов $x_1 \dots x_n$ равна $(1 - \frac{1}{m})^{kn} = e^{-kn/m}$ в силу второго замечательного предела.

Фильтр Блума ошибочно скажет, что некоторый элемент s есть в множестве, если все $h_1(s) \dots h_k(s)$ окажутся равными 1. Вероятность того, что бит $h_1(s)$ окажется равным 1 равна $1 - e^{-kn/m}$, значит вероятность того, что все биты $h_1(s) \dots h_k(s)$ окажутся равным 1 равна $(1 - e^{-kn/m})^k$.

Минимум этой функции при фиксированных n, m равен $\frac{m}{n} \ln 2 = k_{opt}$, сама вероятность ложного срабатывания равна $(1 - e^{-(m/n \ln 2)n/m})^{m/n \ln 2} = 2^{-m/n \ln 2} \approx 2^{-0.693mn} \approx 0.62^{m/n}$, получаем, что при росте размера массива m вероятность уменьшается, а при увеличении числа вставленных элементов вероятность увеличивается, что логично.

2.2 Фильтр кукушки

Фильтр кукушки [6] – вероятностная структура данных, поддерживающая операции добавления, проверки на принадлежность и удаления элемента. В данной работе не рассматривается подробно вероятность получить ложнопо-

ложительный ответ для фильтра кукушки, т.к. это нетривиально. Авторы фильтра кукушки утверждают, что при определенной вероятности получить ложноположительный ответ, фильтру кукушки нужно значительно меньше памяти.

Рассмотрим работу фильтра кукушки. Выбираются две хэш функции $h_1(x),\ h_2(x)$. При добавлении элемента в множество необходимо рассмотреть несколько случаев. Если одно из ячеек $h_1(x),\ h_2(x)$ свободно, то помещаем туда элемент x. Иначе, выбираем случайно который из ячеек необходимо освободить для элемента x. Пусть это будет $h_1(x)$ для удобства. Запоминаем элемент в ячейке $h_1(x)$, вставляем туда элемент x, и запускаем процедуру добавления в множество ранее удаленного элемента. Если такой алгоритм зацикливается, то есть мы приходим к тому, что необходимо удалить элемент, который мы вставляли в множество в процессе, то происходит перестройка всей таблицы. Выбираются новые хэш-функции $h_1(x), h_2(x)$ и заново заполняются все ячейки, то есть все элементы, ранее добавленные в множество добавляются снова в новое пустое множество.

Операция проверки вхождения элемента x в множество – это просто проверка, что элемент x лежит либо в $h_1(x)$, либо в $h_2(x)$.

При операции удаления смотрится, есть ли элемент в ячейках $h_1(x)$ или $h_2(x)$, и если есть, удаляется из этой ячейки.

2.3 Quotient filter

Quotient Filter, Cascade Filter [7] – вероятностная структура данных. Алгоритм работы не разбирался в процессе курсовой работы. Авторы утверждают, что данная структура данных можешь посоревноваться с базами данных, которые упомянуты в главе "1. Введение". Также авторы утверждают, что используется на 20% больше памяти, чем в фильтрах Блума, но при этом работает значительно быстрее. Поддерживаются операции добавления, удаления и проверки на принадлежность множеству.

2.4 Vacuum Filter

Vacuum filter [8] – еще одна вероятностная структура данных, работа которой не рассматривалась в процессе курсовой работы, т.к. из описания алгоритма работа данной структуры кажется очень сложной. Авторы пишут, что эта структура использует на 25% меньше памяти чем фильтр кукушки при такой же скорости работы и вероятности получить ложноположительный ответ, а также на 15% меньше памяти и 10-кратное увеличение скорости по сравнению с фильтром Блума при одинаковой вероятности получить ложноположительный ответ.

3 Реализация в ClickHouse

Для внедрения и выбора лучшего вероятностного фильтра в ClickHouse были созданы классы с самими фильтрами (например ‘BloomFilterBasic‘). В процессе работы возникли сложности с подсчетом k хэш-функций, т.к. на вход фильтра Блума уже приходил результат вычисленной хэш-функции в 64-битном формате. Т.к. генерировать с помощью это функции $h_2(x)$ не имело смысла, было принято решение использовать первые 32 бита и вторые 32 бита отдельно для генерации k хэш-функций. Также отдельно нужно решать задачу, когда ключом фильтра Блума является составной ключ из нескольких колонок. В этом случае нужно быстро вычислять хэш всех объектов. Поэтому сначала вычислялся 64-битный хэш первой колонки, затем второй и результат комбинировался. Это позволило использовать преимущество колоночной СУБД, какой является ClickHouse.

В операцию, которая отвечала за синтаксический анализатор запроса были добавлены классы, которые позволяли использовать вероятностные фильтры (напомним, что в ClickHouse такой возможности не было).

Устройства этих фильтров позволяют рассчитывать на интересные результаты на практике, эксперименты с которыми проводятся в настоящий момент, но ещё не завершены, и поэтому в данной работе не будут представлены ре-

зультаты тестов в ClickHouse. Результаты экспериментов ожидаются на презентации к защите работы.

Работа была выполнена на языке C++ стандарта 2017 года, был оформлен pull request в репозиторий ClickHouse на GitHub. Тесты являлись SQL запросами в ClickHouse.

4 Заключение

В ClickHouse был сделан pull request, где добавляется возможность использовать вероятностные фильтры с некоторыми ограничениями. Также на презентации ожидается получить результаты тестов и выбрать лучший вероятностный фильтр по оптимальности используемой памяти и по скорости работы.

Ссылку на код можно найти в части "Приложения".

5 Возможные улучшение

В рамках текущей работы не пробовались современные вероятностные фильтры – Quotient Filter, Cascade Filter и Vacuum Filter. В будущем стоит реализовать и сравнить на скорость работы эти фильтры и фильтры Блума с фильтром кукушки. Авторы современных вероятностных фильтров утверждают, что их фильтры в разы быстрее и используют меньше памяти при такой же вероятности получить ложноположительный ответ.

6 Список источников

1. СУБД ClickHouse, 2016, https://clickhouse.tech

2. СУБД PostgreSQL, 1996, https://www.postgresql.org/

3. СУБД Vertica, 2005, https://www.vertica.com/

4. СУБД ParAccel, 2005, https://www.actian.com/

5. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422-426, 1970

6. Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than Bloom. In Proc. 10th ACM Int. Conf. Emerging Networking Experiments and Technologies (CoNEXT '14), pages 75-88, 2014. doi:10.1145/2674005. 2674994

7. Bender, Michael A.; Farach-Colton, Martin; Johnson, Rob; Kuszmaul, Bradley C.; Medjedovic, Dzejla; Montes, Pablo; Shetty, Pradeep; Spillane, Richard P.; Zadok, Erez. Don't Thrash: How to Cache your Hash on Flash, 2011

8. Minmei Wang, Mingxun Zhou, Shouqian Shi, Chen Qian. Vacuum Filters: More Space-Efficient and Faster Replacement for Bloom and Cuckoo Filters. PVLDB, 13(2): 197-210, 2019. DOI: https://doi.org/10.14778/3364324.3364333

7 Приложение

1. Ссылка на код: https://github.com/ClickHouse/ClickHouse/pull/11231/files