ВЕРОЯТНОСТНЫЕ СТРУКТУРЫ ДАННЫХ - ФИЛЬТР БЛУМА

Вергунова А.Е., студентка,
Кубанский государственный технологический университет,
г. Краснодар, Россия

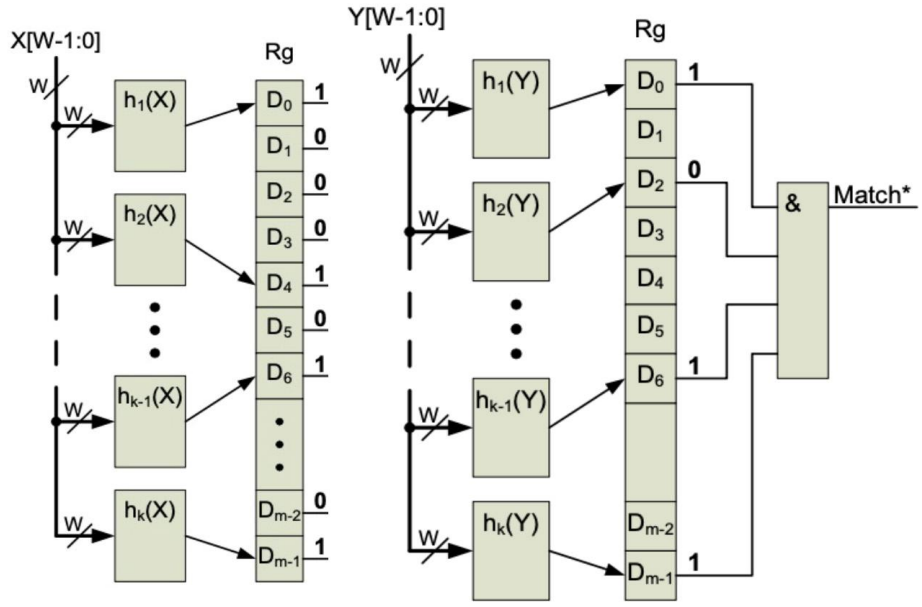
Аннотация: Представлены методы для решения задач хранения и поиска информации на примере фильтра Блума. Приведено описание внутренней структуры фильтра Блума и принципы его работы. Такая структура данных используется в обработке больших объемов данных для эффективного определения принадлежности элемента к множеству, когда задача должна решаться в сжатом размере памяти и быстро. Предложены способы улучшения фильтра, и обсуждаются стратегии для кластеризации и распараллеливание системы, системы с возможностью удаления.

Ключевые слова: большие данные, вероятностные структуры данных, хеш-функция, хранение, фильтр Блума, эффективность, минимизация хранения.

Фильтр Блума – это вероятностная структура данных [1], позволяющая хранить некое множество элементов, а также быстро отвечать на запрос о том, есть ли данный элемент во множестве или нет. При этом существует возможность получить не правильный результат. Если система работы с данными толерантна к незначительным накладным расходам на ошибки, то с помощью фильтра Блума можно значительно повысить ее производительность. Поэтому фильтр Блума нашел применение во многих сферах - в больших данных [2, 3], в дедупликации [4], в сетевых технологиях, включая информационную безопасность [5], файловых системах [2], индексировании и т.п. Фильтр Блума неприменим там, где предъявляются строгие требования к ответам системы, например, базам данных, содержащим данные пользовательской аутентификации (логины, пароли и т.п.).

Построение и реализация фильтра Блума

Фильтр Блума состоит из двух ключевых компонентов (рис. 1): комплекта из k хэш-функций h1(x), h2(x), h3(x), ..., $h_k(x)$ и регистра Rg(массива битов), заполненном нулями в исходном состоянии.



а) Запись б) Проверка вхождения

Рис. 1. Принцип работы фильтра Блума

При записи входное значение подается на вход хэш-функции, которая возвращает индекс 0… m-1, и в данной позиции массива устанавливается «1».

Проверка вхождения проходит аналогично (Рис. 1-б): для фрагмента вычисляются значения всех k хэш-функций. По полученным адресам осуществляется обращение к ячейкам битового массива. Если во всех позициях, на которые укажут хэш-функции, содержатся единицы, считается, что входная комбинация символов с определенной вероятностью совпадает с одним из паттернов, которое уже было записано. Но если хотя бы одна хэш-функция укажет на ячейку с нулевым значением, это гарантированно свидетельствует об отсутствии соответствующей последовательности символов среди множества участвовавших в программировании паттернов.

Если все проверяемые биты равны «1», то нельзя гарантировать, что такие данные уже были. Если был получен неправильный результат — это называется ложноположительное срабатывание (ошибка первого рода) [2].

Ложноотрицательные срабатывания представляют серьезную проблему при операциях удаления, по этой причине большинство реализаций избегают возможности удаления элемента.

Можно предположить, что с увеличением размера m фильтра Блума, число наложений между отпечатками разных ключей должно уменьшаться, что, в свою очередь, приводит к меньшему числу ложных утверждений. Существует ли золотая середина, где и m, и частота ложноположительных срабатываний одновременно малы?

Ошибка первого рода

Значение вероятности ошибки первого рода для заданных значений количества паттернов n, размера битового массива m, и количества хэш-функций k находится согласно формуле (1):

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right) \approx \left(1-e^{-\frac{kn}{m}}\right)^{k} = 1-e^{-\frac{k}{b}} \quad (1),$$

$$\text{где } b = \frac{m}{n} \quad (2)$$

Формула (2) – часто называют пространством b, которое показывает кол-во бит на ключ.

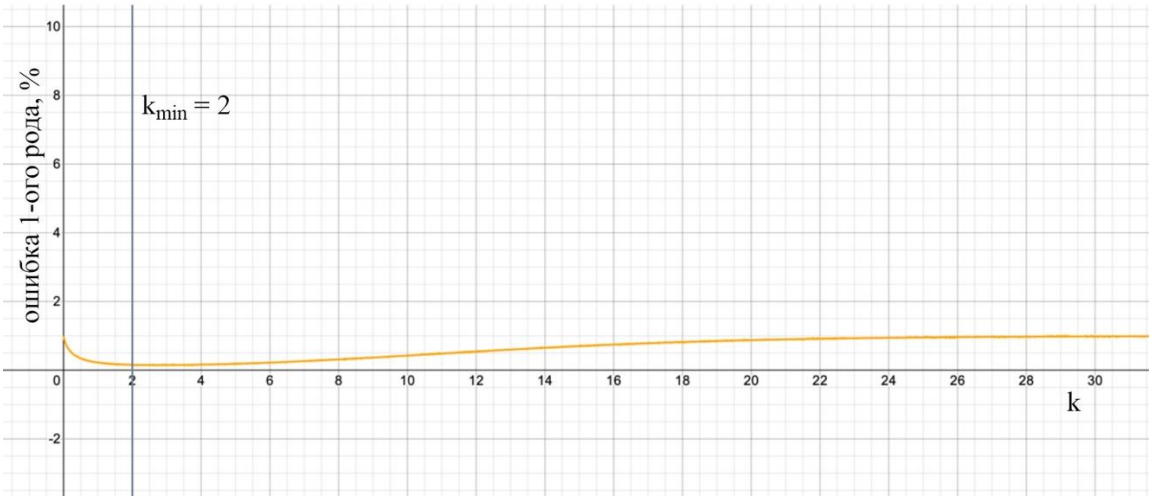


Рис. 2. График по формуле (1) при b = 4, зависимость кол-ва хеш-функций и ошибки 1-го рода

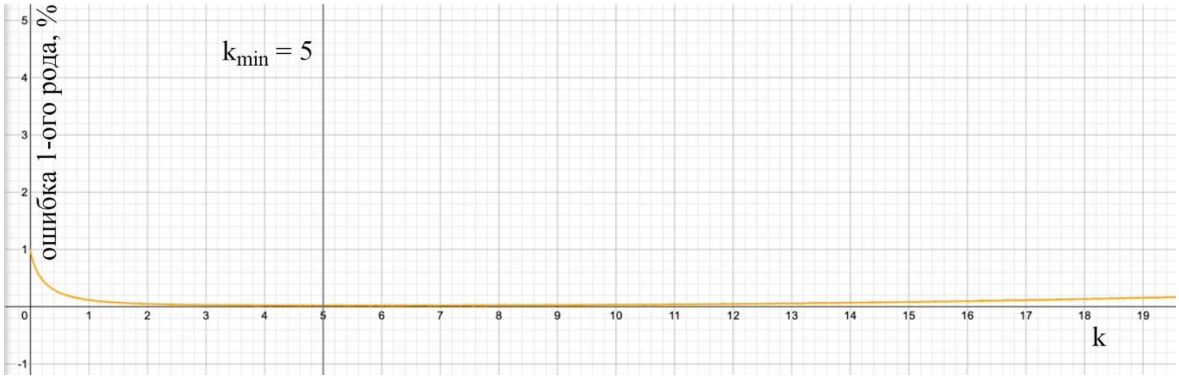


Рис. 3. График по формуле (1) при b = 8, зависимость кол-ва хеш-функций и ошибки 1-го рода

Значение k находится под полным контролем проектировщика фильтра Блума, поэтому мы можем подобрать наилучшее k для минимизации частоты ложных утверждений (Рисунок 2, 3). С помощью дифференциального исчисления получаем:

$$k = \frac{m}{n} \ln(2) = b \times [(\ln] 2) \qquad (3)$$

Тогда вероятность ошибки при оптимальном k:

$$2^{-k} = 2^{-(\ln][2) \times b} \qquad (4)$$

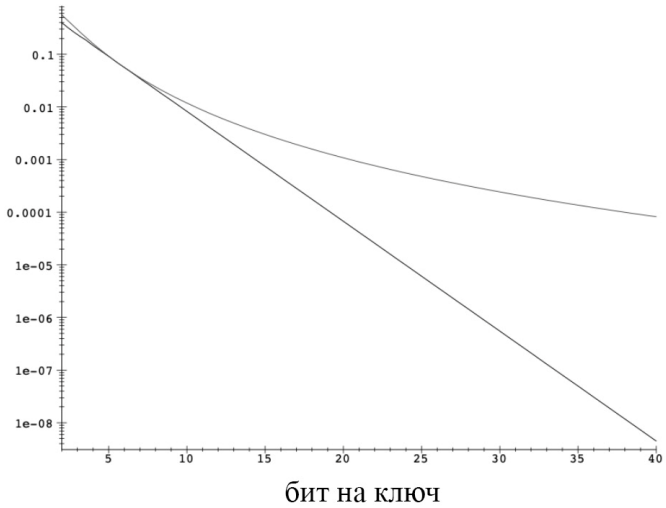


Рис. 4. Вероятность ложноположительных срабатываний от кол-ва бита на ключ. Верхняя кривая – работа фильтра с 4-мя хеш-функциями, нижняя - оптимальное число функций

Таблица 1. Данные, приведенные по формулам (3) и (4)

b, бит	**k**	**ошибка 1-ого рода, %**
4	2,77	14,63415427
8	5,54	2,141584712
16	11,09	0,045863851
24	16,63	0,000982213
32	22,18	2,10349E-05
40	27,72	4,50481E-07
48	33,27	9,64743E-09
56	38,81	2,06608E-10
64	44,36	4,42468E-12
72	49,90	9,47583E-14
80	55,45	2,02933E-15
88	60,99	4,34598E-17
96	66,54	9,30729E-19
104	72,08	1,99323E-20
112	77,63	4,26868E-22
120	83,17	9,14174E-24
128	88,72	1,95778E-25

У нас есть формула, которая выдает ожидаемую частоту ложных утверждений как функцию объема пространства. Данная формула уменьшается экспоненциально вместе с пространством b в расчете на ключ, и именно поэтому существует золотая середина, где и размер фильтра Блума, и его частота ложных утверждений малы одновременно. Например, при хранении только 8 бит на ключ (b = 8) эта оценка составляет чуть более 2 % (Таблица 1). В таблице 1 число k нужно привести к целому числу, для разных вариантов построения фильтра может быть выбран свой способ округления.

Способы улучшения фильтра Блума

Вот несколько способов улучшить фильтр Блума:

Размер фильтра: можно увеличить размер фильтра, чтобы уменьшить вероятность ложноположительных результатов.

Выбор хеш-функций: нужно выбрать такие хеш-функции, которые будут равномерно распределять значения по битам фильтра.

Множественные фильтры: использовать несколько фильтров, в каждом из которых будут свои хеш-функции. Первый способ – каждую функцию привязать

к одному фильтру, каждый новый паттерн обрабатываться каждой хеш-функцией, она устанавливает единицу в свой фильтр, тогда паттерн будет записан во всех фильтрах. Если использовать такой подход, получается, что частота ложноположительных срабатываний будет зависеть только от работы конкретной хеш-функции, так как каждая имеет свой фильтр для записи, но возникают проблемы, нужно увеличивать его размер для избежаний коллизий. Это будет хорошо только для малого числа данных. Второй способ, в отличие от первого, предлагает не записывать каждый паттерн во все фильтры, а разделить паттерн на категории в зависимости от их свойств – длине, значению, или же хешировать для определения категории. Такой способ называется кластеризация, о нем можно подробнее будет рассказано ниже.

Варианты с множественными фильтрами предполагает, что функции могут работать в одной массиве, в котором есть несколько фильтров, тогда результатом работы такой функции будет лежать в диапазоне своего указанного фильтра.

Динамическое изменение размера: динамически изменять размер фильтра в зависимости от количества элементов, которые вы хотите добавить. Это уменьшит расход памяти, когда заранее не известен объем данных.

Подсчет ошибок: Мониторьте и измеряйте вероятность ложноположительных и ложноотрицательных результатов. Это позволит вам оптимизировать параметры фильтра Блума.

Кластеризация

Кластеризация фильтра Блума означает разделение данных на кластеры. Разделение, может быть, по свойствам входящих данных или же использовать нескольких хеш-функций, которые применяются для разделения данных на несколько кластеров.

Вот как это можно реализовать при использовании нескольких хеш-функций:

1. Хеш-функции первого уровня: начните с использования нескольких хеш-функций первого уровня для разделения вашего набора данных на несколько кластеров.

2. Фильтры Блума внутри каждого кластера: для каждого кластера создайте свой собственный фильтр Блума, в котором будут храниться элементы этого кластера. Выберите соответствующий размер и хеш-функции для каждого фильтра Блума.

3. Поиск в кластерах: когда вы хотите проверить, принадлежит ли элемент множеству, сначала примените хеш-функции первого уровня к элементу, чтобы определить кластер, в котором он, возможно, находится. Затем используйте соответствующий фильтр Блума для проверки наличия элемента внутри этого кластера.

Преимущество - помогает уменьшить вероятность ложноположительных результатов, поскольку хеширование на два уровня позволяет более точно разбивать данные. Недостатки - требует больше памяти и вычислительных ресурсов для поддержки множества фильтров Блума.

Распараллеливание

Распараллеливание фильтра Блума — это один из способов улучшения его производительности, особенно в многопоточных или распределенных системах. Способы такой реализации:

Распределенное хранение: использовать распределенную среду, чтобы хранить несколько копий фильтра Блума на разных серверах или узлах, что позволяет параллельно обрабатывать запросы на проверку принадлежности элемента к множеству.

Многопоточное добавление: если у вас есть многозадачное приложение, вы можете использовать несколько потоков для одновременного добавления элементов в фильтр Блума. Это ускорит процесс наполнения фильтра и обеспечит параллельную обработку операций.

Параллельный поиск: вы можете использовать несколько потоков или ядер процессора для одновременного применения хеш-функций и проверки результатов в фильтре Блума.

Важно правильно согласовать доступ к фильтру Блума из разных потоков или узлов, чтобы избежать состояний гонки и других проблем с совместным доступом к данным. Также следует учитывать, что параллельное выполнение может повлечь за собой дополнительные затраты на согласование и управление потоками или узлами.

Фильтр Блума с подсчётом

Впервые подобная модификация была предложена в работе [3] для добавления и удаления элементов в наборе данных.

В отличие от фильтра Блума, фильтр Блума с подсчётом представляет собой m счётчиков с несколькими битами вместо битов. Когда элемент добавляется или удаляется из набора данных, к элементу применяются k хеш-функций и все из k местоположений в массиве увеличиваются или уменьшаются на единицу. Операция поиска проверяет, что каждый из требуемых счётчиков не равен нулю.

Арифметическое переполнение счётчиков при малом кол-ве счетчиков является проблемой, а большие счетчики будут занимать лишнее место, поэтому нужно выбрать оптимальное число счетчиков для системы. Существует реализация под названием «Фильтр Кукушки» [4], имеющая меньшую вероятность ошибки при удалении, но обладающая низкой производительностью.

Литература

1. Burton B.H. Space/time trade-offs in hash coding with allowable errors // Communications of the ACM T. – 1970. – Vol. 13. – No. 7. P. 422–426

7. Fan B., Andersen D. G., Kaminsky M., Mitzenmacher M. D. Cuckoo filter: Practically better than bloom. – New York, NY, USA: ACM – 2014. – P. 75–88.

2. Chang F., Dean J., Ghemawat S., Hsieh W.C., Wallach D.A., Burrows M., Chandra T., Fikes A., Gruber R.E. A distributed storage system for structured data // ACM Transactions on Computer Systems. – 2008. – Article. 4. – P. 26.

6. Fan L., Cao P., Almeida J., Broder A. Summary cache: A scalable wide-area Web cache sharing protocol. – IEEE/ACM Transactions on Networking. – 2000. – Vol. 8. – No. 3. – P. 281–293.

5. Patgiri R., Nayak S., Borgohain S. K. Preventing DDoS using bloom filter: A survey // EAI Endorsed Transaction on Scalable Information Systems. – 2018. – Vol. 13. – P. e3.

3. Patgiri R., Nayak S., Borgohain S. K. Role of Bloom Filter in Big Data Research: A Survey // International Journal of Advanced Computer Science and Applications(IJACSA). – 2018.– Vol. 9. – No 11. – P. 655–661.

4. Zhang Y., Wu Y. and Yang G. Droplet: A distributed solution of data deduplication // ACM/IEEE 13th International Conference on Grid Computing. – 2012. – P. 114–121.