УДК 004

РАЗРАБОТКА МОДУЛЯ ПРОВЕРКИ ДАННЫХ ДЛЯ УДОВЛЕТВОРЕНИЯ МЕТРИКИ УСТАРЕВАНИЯ

А. И. Сибгатуллина[1][0000-0003-4014-9558], **А. Ш. Якупов**[2][0000-0002-2333-8819]

[1, 2]Казанский (Приволжский) федеральный университет, ул. Кремлевская, 35, г. Казань, 420008

[1]aigul.sibgatulli@gmail.com, [2]azat.yakupov@it.kfu.ru

Аннотация

Из года в год возрастает объем мирового рынка больших данных. Их анализ является неотъемлемой частью для принятия немедленных и надежных решений. Технологии больших данных ведут к значительному снижению стоимости за счет использования облачных сервисов, распределенных файловых систем, когда возникает потребность в хранении больших объемов информации. Их аналитика неразрывно связана с понятием качества данных, что особенно важно, если они имеют определенный срок хранения – метрику устаревания – и мигрируют из одного источника в другой, увеличивая риск потери данных. Предупреждение негативных последствий достигается за счет процесса сверки данных – комплексной проверки больших объемов информации с целью подтверждения их согласованности.

В статье рассмотрены вероятностные структуры данных, которые могут быть использованы для решения задачи, а также предложена реализация – модуль проверки целостности данных с использованием фильтра Блума с подсчетом. Данный модуль интегрирован в Apache Airflow для автоматизации процесса.

***Ключевые слова**: большие данные, метрика устаревания, партиция, parquet файл, фильтр Блума*

ВВЕДЕНИЕ

В эпоху больших данных каждая компания ежедневно обрабатывает огромные объемы информации. Согласно исследованию, опубликованному компанией MarketsandMarkets [1], мировой объем рынка Big Data оценивается в 162,6 миллиарда долларов по итогам 2021 года, однако прогнозируется, что к 2026 году он

© А. И. Сибгатуллина, А. Ш. Якупов, 2022.
Данная статья распространяется на условиях международной лицензии Creative Commons License Attribution 4.0 International (CC BY 4.0).

вырастет до 273,4 миллиарда долларов при средней динамике в 11,9% в год. Одними из преимуществ аналитики больших данных являются скорость и эффективность. Если всего несколько лет назад предприятия собирали и извлекали информацию, проводили аналитику для принятия только будущих решений, то сегодня этот процесс ориентирован на немедленные и более обоснованные решения, что предоставляет организациям конкурентное преимущество: с одной стороны, возможность работать быстрее, с другой – сохранять гибкость.

Аналитика больших данных неразрывно связана с такими понятиями, как управление данными и качество данных. Перед проведением анализа необходимо удостовериться в их целостности, что особенно важно во время проведения миграции, так как существует высокий риск потери части данных вследствие таких факторов, как: низкая пропускная способность сети или ее отключение, нестабильное соединение, прерывание транзакций, сбои во время выполнения заданий, выход из строя узлов кластера. Все перечисленные ошибки могут привести к тому, что данные будут сохранены в недопустимом состоянии, например, содержать неверные или некорректно сформатированные значения, дублирующиеся строки, или, наоборот, часть информации будет потеряна.

Кроме того, многие компании [2] руководствуются политикой хранения данных (или записей), которая является ключевой составляющей жизненного цикла информации. Она представляет собой установленный в организации протокол и описывает, как долго необходимо хранить данные, в каком месте и каким образом их уничтожить по истечении срока использования. Эта политика важна вследствие ряда причин. Во-первых, она снижает затраты на хранение ненужных данных, начиная с вопросов места и памяти и заканчивая экономическими аспектами. Во-вторых, повышает релевантность существующих данных, так как ранняя информация становится менее актуальной по мере устаревания. Однако, если удаление данных внутри организации не представляется возможным, две вышеупомянутые причины становятся проблемами, которые необходимо решить.

Наиболее распространенным способом является использование «холодного хранилища» (например, Hadoop HDFS, Amazon S3) для длительного содержания объектов с редкими запросами на чтение. По истечении срока использования часть нерелевантных данных из СУБД перемещается в холодное хранилище.

При необходимости хранить данные на протяжении длительного периода возрастает важность обеспечения их качества. Если после истечения срока, установленного политикой хранения, обнаружится, что мигрированные данные повреждены, то они будут безвозвратно утеряны. Предупреждение негативных последствий достигается за счет процесса сверки данных – комплексной проверки больших объемов информации с целью подтверждения их согласованности. Во время него происходит сравнение исходных данных с целевыми для определения стабильности архитектуры миграции.

Так как это область больших данных, использование стандартных методов, например, построчного сравнения, неприменимо, поэтому становится актуальной проблема отсутствия автоматизированного процесса проверки качества больших данных.

ОБЗОР ЛИТЕРАТУРЫ

Вероятностные структуры данных

Неограниченный рост данных привел к смене парадигмы в методах хранения и поиска от традиционных структур данных к вероятностным. Детерминированные структуры всегда дают точные ответы и, как и вероятностные, могут выполнять то же множество операций, но только с малыми наборами данных. Если размер датасета велик и не помещается в память, то детерминированные структуры дают сбой, и их использование не представляется возможным. Вероятностные, в свою очередь, подходят для работы с большими данными и потоковыми приложениями, так как позволяют избежать высокой задержки аналитических процессов. Эти структуры используют хеш-функции для компактного представления набора элементов. Они не могут дать определенного ответа, только приближенный, но по сравнению с детерминистическими структурами они требуют гораздо меньше памяти и имеют постоянное время обработки сложных запросов.

Существует несколько типов вероятностных структур данных, изображенных на рисунке 1, которые решают следующие задачи:

- проверка на членство в множестве;
- подсчет частоты;

- оценка кардинальности – подсчет количества раз, когда элемент встречался в массивных наборах данных;
- поиск сходства – поиск ближайших соседей в датасете.

Рисунок 1. Вероятностные структуры данных

В статье [3] обсуждаются различные сферы применения подобных структур. Например, Bloom Filter изначально создавался с целью представления слов в словаре. Постепенно он стал широко использоваться в сетевых алгоритмах и алгоритмах безопасности, таких как аутентификация, отслеживание IP-адресов, поиск подстроки. Кроме того, сотовые сети обеспечивают связь между устройствами с применением фильтра Блума для идентификации мобильных приложений [4]. Другим примером является метод MinHash, позволяющий найти сходства между двумя элементами, вычисляя коэффициент Жаккара. В настоящий момент она используется в различных областях: при кластеризации изображений для поиска дубликатов [5], для обнаружения вредоносных программ [6].

В [3] также экспериментально подтверждено, что асимптотическая сложность вероятностных структур данных гораздо меньше, чем детерминистических при выполнении операций вставки, деления, обхода, поиска наряду с другими статическими запросами. Следовательно, учитывая экспоненциальный рост данных и доменных областей, такие структуры позволяют ускорить процессы по обработке информации.

Для решения задачи принадлежности элемента множеству две структуры из обозначенных на рисунке 1 могут быть использованы в качестве его основы: Bloom Filter и Quotient Filter.

Несмотря на то что оба подхода поддерживают один набор операций и одну и ту же сложность, Quotient Filter имеет некоторую вероятность ложноотрицательного срабатывания [7], то есть при наличии элемента в множестве возвращать результат об его отсутствии, что несвойственно для Bloom Filter. Помимо этого, он использует больше памяти, но по скорости сравним с фильтром Блума. Другим недостатком данной структуры является резкое снижение производительности более, чем в два раза, по мере ее заполнения. Временные расходы происходят из-за комплексной процедуры хеширования [8]: нахождение подходящей позиции для элемента при вставке со сдвигом является трудоемкой задачей ввиду возможности коллизий. Также в [8] проиллюстрирован сравнительный анализ частоты коллизий в зависимости от числа хеш-функций. Изначально Bloom Filter с меньшим набором функций имеет большее число коллизий по сравнению с Quotient Filter, однако оно значительно уменьшается с увеличением количества функций.

Имплементации фильтра Блума

На текущий момент существует более 60 различных вариаций фильтра Блума [9], десятки из которых посвящены сокращению ложноположительных срабатываний и упрощению реализации с целью улучшения производительности алгоритма. Их сравнение производилось на основе нескольких параметров:

- подсчет – подсчет количества использованных битов, который может быть осуществлен как напрямую, так и с помощью набора хеш-функций или других альтернативных решений;
- группировка – нахождение подмножества, к которому принадлежит определенный элемент;
- удаление и масштабируемость;
- распад – исключение устаревших элементов для фокусировки на более новых;
- параллелизм – параллельные вычисления, обеспечивающие ускорение запроса и увеличение его пропускной способности;
- ложноотрицательное срабатывание.

Стоит отметить, что ни один из рассмотренных вариантов не удовлетворяет всем вышеописанным характеристикам, при этом треть из них имеет сложность

вставки и запроса выше, чем стандартная реализация. Например, для Bloomier Filter [10] вследствие того, что он рекурсивно кодирует все элементы, применяя несколько фильтров Блума, сложность вставки составляет $O(n\,logn)$, а запроса – $O(\lambda k)$, где λ – количество фильтров Блума. Кроме того, многие из подтипов применимы только к определенным доменным областям, а именно: сетевое взаимодействие (Adaptive BF [11], Energy efficient BF [12]), обработка биометрических (k-mer BF [13]) и пространственных (Spatial BF [14]) данных, системы хранения (BloomStore [15]), дублирование (Stable BF [16]) и обнаружение копий (Matrix BF [17]). А также 15 типов из представленных 60 вводят ложноотрицательное срабатывание, которое отсутствует в оригинальном фильтре Блума.

В статье [18] предложена Persistent Bloom Filter (PBF) – вероятностная структура, которая поддерживает тестирование временного членства в множестве. Под временным диапазоном понимается разница между двумя моментами времени для каждого элемента. В качестве примера для тестирования используются IP-адреса: например, пользователь мог войти сеть несколько раз в течение одного заданного интервала, поэтому дубликаты допустимы. Таким образом, цель данного тестирования – обнаружить строки или столбцы с определенными значениями, которые были добавлены или изменены в течение указанного периода. Решение данной проблемы – PBF – представляет собой цепочку фильтров Блума, где каждый из них ответственен за отобранное подмножество элементов. Он декомпозирует запрос и отправляет отдельные части разным фильтрам. К его особенностям относятся: отсутствие ложноотрицательных срабатываний, эффективное использование памяти, производительное обновление данных и высокая точность за счет регулирования общего количества битов и для каждого отдельного фильтра.

Одним из вариаций фильтра Блума является также фильтр Блума с подсчетом (Counting Bloom Filter). Он поддерживает операцию удаления благодаря использованию счетчика, при этом не вводит ложноотрицательное срабатывание [19]. Эта функциональность критически важна, когда предоставляемый набор данных динамичен, то есть его размер может изменяться со временем. Кроме того, описываемый фильтр использует меньший объем памяти по сравнению со

стандартным подходом, но потребляет больший при хранении отпечатка каждого элемента.

Для решения поставленных задач нами выбран фильтр Блума с подсчетом в качестве основы для реализации алгоритма вследствие следующих причин. Во-первых, он имеет низкий уровень расходов по времени, памяти и вычислениям [3]. Во-вторых, он является структурой общего назначения [9], то есть не относится к конкретной предметной области и может быть использован в совокупности с любыми форматами входных данных. В-третьих, позволяет хранить повторную информацию и исключать элементы из множества. В-четвертых, в фильтре Блума с подсчетом точное значение счетчика не учитывается помимо того, положительное оно или отрицательное [20], что предоставляет достаточный результат для определения принадлежности к множеству.

ИСПОЛЬЗУЕМЫЕ ФОРМАТЫ И СТРУКТУРЫ ДАННЫХ

Партиционная таблица

Партиционирование (или секционирование) – это разделение хранимых объектов баз данных (таблиц, индексов, материализованных представлений) на более мелкие логические части по заданным критериям. Партиционная таблица представляет собой специального вида таблицу, которая поделена на сегменты, называемые партициями, для того, чтобы обеспечить более удобное и быстрое управление данными. Физически партиции могут находиться в одном табличном пространстве, в разных или комбинируя оба подхода. За счет разбиения большой таблицы на более мелкие секции повышается производительность запросов, так как обращение идет только к части данных, и уменьшаются расходы на память, что является ключевым фактором при работе в области больших данных. Таким образом, основная цель партиционирования – это помощь в обслуживании объемных таблиц и сокращение общего времени отклика на чтение и загрузку данных для определенных SQL операций.

Apache Parquet

Apache Parquet представляет собой бинарный, колоночно-ориентированный формат хранения больших данных. Он предоставляет возможности задавать

схемы сжатия на уровне столбцов и добавлять новые кодировки. Изначально созданный для экосистемы Hadoop, он является одним из наиболее распространенных форматов Big Data наряду с Apache Avro, RCFile и ORC.

Parquet впервые появился в 2013 году, и с тех пор получил широкое распространение в качестве бесплатного формата хранения данных с открытым исходным кодом, ориентированного для быстрого выполнения аналитических запросов. Когда компания AWS анонсировала [21] функциональность экспорта озер данных, она охарактеризовала его как формат, позволяющий выгружать в два раза быстрее и использующий в шесть раз меньше места в хранилище Amazon S3 по сравнению с текстовыми форматами.

Структура Parquet файла проиллюстрирована на рисунке 2, в которой можно выделить три составляющие:

1. Группа строк (row group);
2. Фрагмент столбца (column chunk);
3. Страница (page).

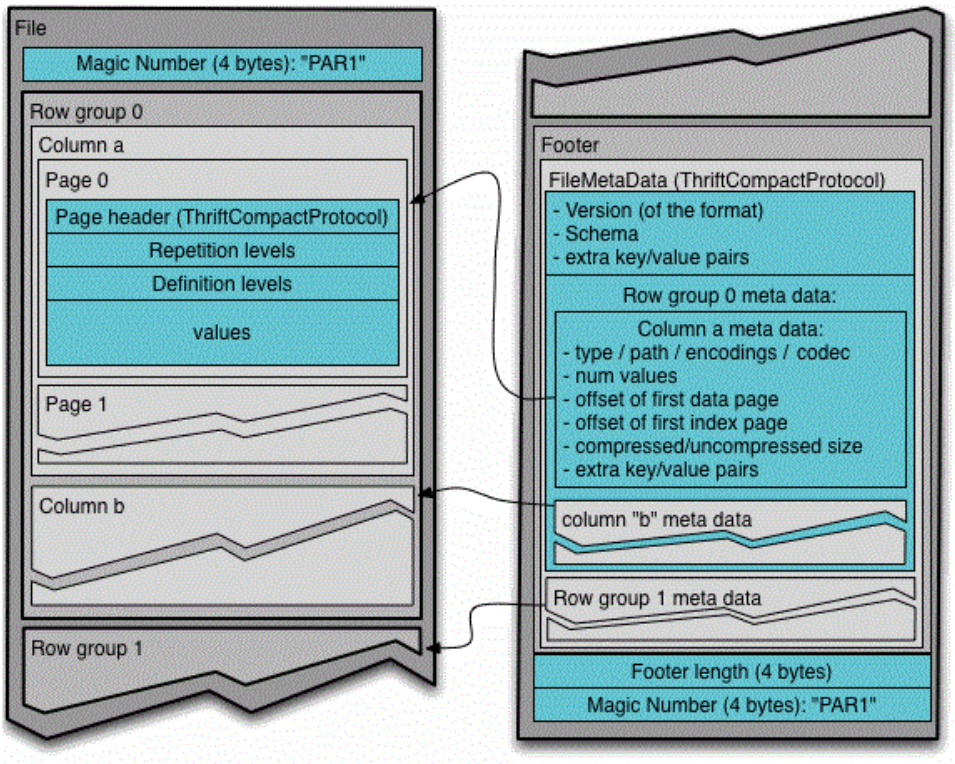

Рисунок 2. Структура файла формата Apache Parquet

Группа строк представляет собой набор строк в колоночно-ориентированном формате размером от 50 мегабайт до 1 гигабайта. Она состоит из фрагмента каждого столбца в наборе данных. Фрагмент столбца – это блок данных конкретного столбца в определенной группе строк. Страница – это концептуально неделимая единица, которая содержит фрагменты столбцов. Страницы записаны друг за другом, содержат метаинформацию и закодированные данные, поэтому при необходимости можно считать только определенные страницы.

В отличие от линейно-ориентированных форматов, parquet оптимизирован для повышения производительности. При выполнении запросов фокусировка происходит только на необходимых данных. Кроме того, объем сканируемой информации сводится к минимуму, что ведет к рациональному использованию операций ввода-вывода. В одном из практических экспериментов [22], проведенном американской компанией Databricks, было подтверждено, что использование формата parquet сократило расходы на память как минимум на одну треть для больших объемов данных, а также уменьшило временные затраты на сканирование и десериализацию более, чем в 34 раза. Таким образом, при работе с большими данными Apache Parquet является одним из наиболее подходящих форматов, позволяющим оптимизировать время запросов и увеличить производительность.

Фильтр Блума

Фильтр Блума – это эффективная вероятностная структура данных, созданная Бертоном Блумом в 1970 году, которая определяет, является ли элемент частью множества. Он предотвращает выполнение лишних вычислений, проверяя тот факт, что элемент совершенно точно не входит в множество. В отличие от линейного и бинарного поисков, сложности которых составляют $O(n)$ и $O(\log n)$ соответственно, сложность вставки и проверки принадлежности к множеству с помощью фильтра Блума – $O(1)$. Он никогда не выдаст ложноотрицательные результаты, но, вследствие того, что структура является вероятностной, в этом подходе существует возможность получения ложноположительных (false positive) результатов. Под ложноположительным (ложноотрицательным) срабатыванием будет понимать положительный (отрицательный) ответ структуры данных при отсутствии (наличии) в ней элемента. Другими словами, все элементы множества

распознаются корректно, однако есть вероятность получить положительный итог при отсутствии значения.

Для описания работы фильтра Блума необходимо ввести несколько переменных. На начальном этапе пустой фильтр представляет собой битовый массив из m битов, равных 0:

	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
m	0	1	2	3	4	5	6	7	8	9

Под n будем считать количество элементов множества S. Фильтр Блума основан на хешировании, поэтому количество хеш-функций обозначим k. Каждая функция сопоставляет элементы с корзинами (бакетами) соответственно битовому массиву. При добавлении элемента e в фильтр рассчитываются значения всех хеш-функций, затем индексы при помощи оператора деления по модулю, которые заменяются на 1 в битовом массиве:

$$\forall e: h_1(e), \dots, h_k(e) => S = \{S_1, S_2, \dots, S_m\} \cup \{e\}$$

Например, для элемента e с применением двух хеш-функций верно:

$$h_1(e)\ \%\ m = 2,$$
$$h_2(e)\ \%\ m = 6,$$

	0	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
m	0	1	2	3	4	5	6	7	8	9

Для осуществления проверки необходимо пройти через весь процесс в обратном порядке: рассчитать результаты хеш-функций для входного значения и посмотреть, все ли индексы битового массива равны 1. В случае, если все биты содержат 1, элемент либо точно существует в множестве, либо отсутствует ввиду ложноположительного срабатывания. Если хотя бы один из битов равен 0, то можно утверждать, что элемент отсутствует:

$$\forall e: h_1(e), \dots, h_k(e);\ \forall i = \overline{1,n};\ \exists BF_i = 1 => \{e\} \in S$$
$$\forall e: h_1(e), \dots, h_k(e);\ \forall i = \overline{1,n};\ \exists BF_i = 1 => false\ positive$$
$$\forall e: h_1(e), \dots, h_k(e);\ \forall i = \overline{1,n};\ \exists BF_i = 0 => \{e\} \notin S$$

Ложноположительный результат может возникнуть тогда, когда все биты входного значения установлены 1 при добавлении других элементов. Его вероятность можно контролировать путем изменения размеров фильтра Блума: чем больше битовый массив, тем ниже вероятность ложных срабатываний. Кроме

того, увеличение количества хеш-функций также ведет к уменьшению вероятности, однако добавляет временную задержку при дополнении и поиске.

Фильтр Блума поддерживает две операции – вставку и проверку – но не позволяет произвести удаление элемента. Если изменять биты в массиве, то это может привести к ложноотрицательным результатам. Наиболее популярным расширением классического фильтра Блума является фильтр Блума с подсчетом. Он вводит массив из m счетчиков, соответствующий каждому биту в массиве.

Фильтр Блума с подсчетом позволяет приблизительно определить, сколько раз каждый элемент был отмечен в фильтре, увеличивая счетчик при добавлении нового элемента. При этой операции сначала вычисляются соответствующие ему битовые позиции, а затем для каждой из них инкрементируется счетчик. Благодаря этому при удалении элемента возможны модификации массива, так как его значения всегда будут неотрицательными. Нами использовано данное расширение ввиду того, что оно дает возможность хранить повторную информацию и не теряет своей функциональности при удалении.

РЕЗУЛЬТАТЫ

После изучения различных вероятностных структур, которые могут быть использованы для определения принадлежности элемента множеству, был разработан и протестирован модуль проверки партиции и parquet файла на основе фильтра Блума с подсчетом, затем он был интегрирован в процесс в Apache Airflow.

Процесс запускается ежедневно по расписанию и проверяет, появилась ли новая партиция, у которой истек срок хранения. Он задается в днях в отдельной базе данных на уровне партиционной таблицы. Каждая партиция содержит информацию за отдельный месяц. Работа процесса осуществляется в несколько этапов:

1. Формируется список всех партиций, которые в текущий момент хранятся в базе данных;
2. Запрашивается метрика устаревания таблицы. Каждая партиция из списка, созданного на предыдущем шаге, проверяется на соответствие метрике. Если разница между максимальным значением даты и текущей датой превышает допустимый порог, то такая партиция считается устаревшей;

3. Для найденной партиции запрашиваются маппинг столбцов с их типами и сами данные;
4. Считывается файл формата parquet из HDFS кластера;
5. Инициализируется фильтр Блума. В качестве n подается на вход количество строк в партиции. Вероятность P задается вручную в конфигурационном файле. Ориентиром в этом случае служит партиция – она всегда содержит актуальную неискаженную информацию, в то время как паркетный файл может иметь различного рода аномалии: замещение слепков данных, неверное копирование – в связи с нестабильностью сетевого трафика;
6. При успешной проверке партиция отсоединяется и удаляется из базы данных, так как соответствие с parquet файлом подтверждено;
7. При неуспешной проверке либо отсутствии или повреждении файла по заданному пути отправляется информационное сообщение в телеграм-чат.

Полный граф процесса изображен на рисунке 3:

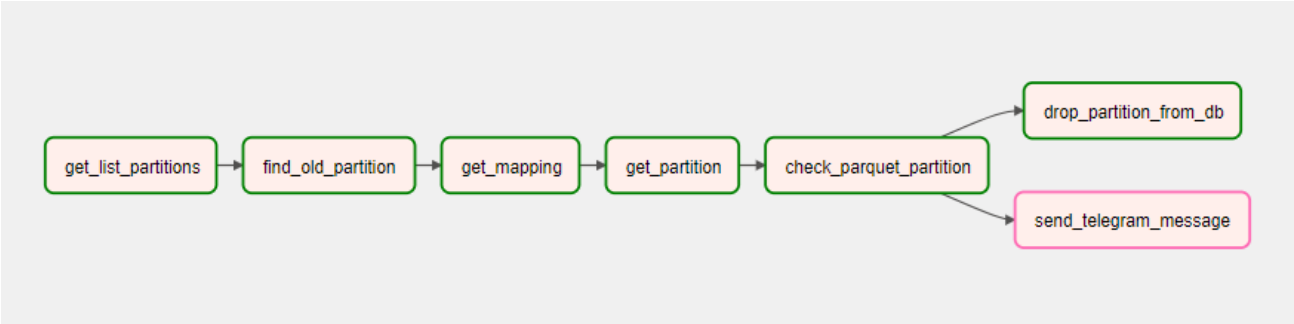

Рисунок 3. Граф процесса проверки parquet файла с данными партиции

ЗАКЛЮЧЕНИЕ

Таким образом, проведенное исследование показало, что на практике для решения задачи проверки целостности файлов с данными могут использоваться вероятностные структуры, определяющие принадлежность элемента множеству. Кроме того, были рассмотрены различные вариации фильтра Блума и реализован наиболее подходящий под критерии задачи.

Результатом исследования является разработка модуля проверки данных партиции и parquet файла на основе фильтра Блума с подсчетом. При разработке процесса использованы: инструмент для создания, планирования и мониторинга рабочих процессов Apache Airflow, распределенная файловая система Apache Hadoop и база данных PostgreSQL.

В дальнейшем планируются подключение брокеров сообщений для получения информации из HDFS кластера, а также применение процесса на основе реальных данных.

СПИСОК ЛИТЕРАТУРЫ

1. Big Data Market worth $273.4 billion by 2026. URL: https://www.marketsandmarkets.com/Market-Reports/big-data-market-1068.html.

2. Data Retention Policy: What Is It and How to Build One. URL: https://www.techtarget.com/searchdatabackup/definition/data-retention-policy.

3. *Batra S., Garg S., Kaur R., Kumar N., Singh A., Zomaya A.Y.* Probabilistic data structures for big data analytics: A comprehensive review // Knowledge-Based Systems. 2019. Vol. 188. No. 104987. P. 54–75.

4. *Choi K.W., Hossain E., Wiriaatmadja D.T.* Discovering mobile applications in cellular device-to-device communications: Hash function and bloom filter-based approach // IEEE Transactions on Mobile Computing. 2016. Vol. 15. No. 2. P. 336–349.

5. *Sasikala J., Thaiyalnayaki S.* Indexing near-duplicate images in web search using minhash algorithm // International Conference on Processing of Materials, Minerals and Energy. 2018. Vol. 5. No. 1. P. 1943–1949.

6. *Drew J., Hahsler M., Moore T.* Polymorphic Malware Detection Using Sequence Classification Methods // IEEE Security and Privacy Workshops (SPW). 2016. P. 81–87.

7. *Borgohain S.K., Nayak S., Patgiri R.* rDBF: A r-Dimensional Bloom Filter for massive scale membership query // Journal of Network and Computer Applications. 2019. Vol. 136. P. 100–113.

8. *Batra S., Garg S., Kumar N., Singh A.* Probabilistic data structure-based community detection and storage scheme in online social networks // Future Generation Computer Systems. 2019. Vol. 94. P. 173–184.

9. *Guo D., Luo L., Luo X., Ma R. T. B., Rottenstreich O.* Optimizing Bloom Filter: Challenges, Solutions, and Comparisons // IEEE Communications Surveys & Tutorials. 2019. Vol. 21. No. 2. P. 1912–1949.

10. *Boy O., Chazelle B., Kilian J., Rubinfeld R., Tal A.* The Bloomier filter: An efficient data structure for static support lookup tables // SODA. 2004. P. 30–39.

11. *Hazeyama H., Kadobayashi Y., Matsumoto Y.* Adaptive Bloom filter: A space-efficient counting algorithm for unpredictable network traffic // IEICE Transactions on Information and Systems. 2008. Vol. 91. No. 5. P. 1292–1299.

12. *Song T., Wang X., Zhou Y.* EABF: Energy efficient self-adaptive Bloom filter for network packet processing // IEEE International Conference on Communications (ICC). 2012. P. 2729–2734.

13. *Filippova D., Kingsford C., Pellow D.* Improving Bloom filter performance on sequence data using k-mer Bloom filters // J. Comput. Biol. 2017. Vol. 26. No. 6. P. 547–557.

14. *Calderoni L., Maio D., Palmieri P.* Location privacy without mutual trust: The spatial Bloom filter // Computer Communications. 2015. Vol. 68. P. 4–12.

15. *Du D.H.C., Lu G., Nam Y.J.* BloomStore: Bloom filter based memory-efficient key-value store for indexing of data de-duplication on flash // IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST). 2012. P. 1–11.

16. *Deng F., Rafiei D.* Approximately detecting duplicates for streaming data using stable Bloom filters // ACM SIGMOD international conference on Management of data. 2006. P. 25–36.

17. *Ahmadi M., Geravand S.* A novel adjustable matrix Bloom filterbased copy detection system for digital libraries // IEEE 11th International Conference on Computer and Information Technology. 2011. P. 518–525.

18. *Guo J., Li F., Peng Y., Qian W., Zhou A.* Persistent Bloom Filter: Membership Testing for the Entire History // International Conference on Management of Data. 2018. P. 1037–1052.

19. *Nayak S., Patgiri R.* A Review on Role of Bloom Filter on DNA Assembly // IEEE Access. 2019. Vol. 7. P. 66939–66954.

20. *Reviriego P., Rottenstreich O.* The Tandem Counting Bloom Filter – It Takes Two Counters to Tango // IEEE/ACM Transactions on Networking. 2019. Vol. 27. No. 6. P. 2252–2265.

21. Announcing Amazon Redshift data lake export: share data in Apache Parquet format. URL: https://aws.amazon.com/about-aws/whats-new/2019/12/announcing-amazon-redshift-data-lake-export/#:~:text=The%20Parquet%20format%20is%20up,lake%20in%20an%20open%20format.

22. Parquet. URL: https://databricks.com/glossary/what-is-parquet.

DEVELOPMENT A DATA VALIDATION MODULE TO SATISFY THE RETENTION POLICY METRIC

Aigul Sibgatullina[1][0000-0003-4014-9558], **Azat Yakupov**[2][0000-0002-2333-8819]

[1, 2]*Kazan (Volga Region) Federal University, 35 Kremlevskaya str., Kazan, 420008*

[1]aigul.sibgatulli@gmail.com, [2]azat.yakupov@it.kfu.ru

Abstract

Every year the size of the global big data market is growing. Analysing these data is essential for good decision-making. Big data technologies lead to a significant cost reduction with use of cloud services, distributed file systems, when there is a need to store large amounts of information. The quality of data analytics is dependent on the quality of the data themselves. This is especially important if the data has a retention policy and migrates from one source to another, increasing the risk of a data loss. Prevention of negative consequences from data migration is achieved through the process of data reconciliation – a comprehensive verification of large amounts of information in order to confirm their consistency.

This article discusses probabilistic data structures that can be used to solve the problem, and suggests an implementation – data integrity verification module using a Counting Bloom filter. This module is integrated into Apache Airflow to automate its invocation.

Keywords: *big data, retention policy, partition, parquet file, Bloom filter*

REFERENCES

1. Big Data Market worth $273.4 billion by 2026. URL: https://www.marketsandmarkets.com/Market-Reports/big-data-market-1068.html.

2. Data Retention Policy: What Is It and How to Build One. URL: https://www.techtarget.com/searchdatabackup/definition/data-retention-policy.

3. *Batra S., Garg S., Kaur R., Kumar N., Singh A., Zomaya A.Y.* Probabilistic data structures for big data analytics: A comprehensive review // Knowledge-Based Systems. 2019. Vol. 188. No. 104987. P. 54–75.

4. *Choi K.W., Hossain E., Wiriaatmadja D.T.* Discovering mobile applications in cellular device-to-device communications: Hash function and bloom filter-based approach // IEEE Transactions on Mobile Computing. 2016. Vol. 15. No. 2. P. 336–349.

5. *Sasikala J., Thaiyalnayaki S.* Indexing near-duplicate images in web search using minhash algorithm // International Conference on Processing of Materials, Minerals and Energy. 2018. Vol. 5. No. 1. P. 1943–1949.

6. *Drew J., Hahsler M., Moore T.* Polymorphic Malware Detection Using Sequence Classification Methods // IEEE Security and Privacy Workshops (SPW). 2016. P. 81–87.

7. *Borgohain S.K., Nayak S., Patgiri R.* rDBF: A r-Dimensional Bloom Filter for massive scale membership query // Journal of Network and Computer Applications. 2019. Vol. 136. P. 100–113.

8. *Batra S., Garg S., Kumar N., Singh A.* Probabilistic data structure-based community detection and storage scheme in online social networks // Future Generation Computer Systems. 2019. Vol. 94. P. 173–184.

9. *Guo D., Luo L., Luo X., Ma R. T. B., Rottenstreich O.* Optimizing Bloom Filter: Challenges, Solutions, and Comparisons // IEEE Communications Surveys & Tutorials. 2019. Vol. 21. No. 2. P. 1912–1949.

10. *Boy O., Chazelle B., Kilian J., Rubinfeld R., Tal A.* The Bloomier filter: An efficient data structure for static support lookup tables // SODA. 2004. P. 30–39.

11. *Hazeyama H., Kadobayashi Y., Matsumoto Y.* Adaptive Bloom filter: A space-efficient counting algorithm for unpredictable network traffic // IEICE Transactions on Information and Systems. 2008. Vol. 91. No. 5. P. 1292–1299.

12. *Song T., Wang X., Zhou Y.* EABF: Energy efficient self-adaptive Bloom filter for network packet processing // IEEE International Conference on Communications (ICC). 2012. P. 2729–2734.

13. *Filippova D., Kingsford C., Pellow D.* Improving Bloom filter performance on sequence data using k-mer Bloom filters // J. Comput. Biol. 2017. Vol. 26. No. 6. P. 547–557.

14. *Calderoni L., Maio D., Palmieri P.* Location privacy without mutual trust: The spatial Bloom filter // Computer Communications. 2015. Vol. 68. P. 4–12.

15. *Du D.H.C., Lu G., Nam Y.J.* BloomStore: Bloom filter based memory-efficient key-value store for indexing of data de-duplication on flash // IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST). 2012. P. 1–11.

16. *Deng F., Rafiei D.* Approximately detecting duplicates for streaming data using stable Bloom filters // ACM SIGMOD international conference on Management of data. 2006. P. 25–36.

17. *Ahmadi M., Geravand S.* A novel adjustable matrix Bloom filterbased copy detection system for digital libraries // IEEE 11th International Conference on Computer and Information Technology. 2011. P. 518–525.

18. *Guo J., Li F., Peng Y., Qian W., Zhou A.* Persistent Bloom Filter: Membership Testing for the Entire History // International Conference on Management of Data. 2018. P. 1037–1052.

19. *Nayak S., Patgiri R.* A Review on Role of Bloom Filter on DNA Assembly // IEEE Access. 2019. Vol. 7. P. 66939–66954.

20. *Reviriego P., Rottenstreich O.* The Tandem Counting Bloom Filter – It Takes Two Counters to Tango // IEEE/ACM Transactions on Networking. 2019. Vol. 27. No. 6. P. 2252–2265.

21. Announcing Amazon Redshift data lake export: share data in Apache Parquet format. URL: https://aws.amazon.com/about-aws/whats-new/2019/12/announcing-amazon-redshift-data-lake-export/#:~:text=The%20Parquet%20format%20is%20up,lake%20in%20an%20open%20format.

22. Parquet. URL: https://databricks.com/glossary/what-is-parquet.

СВЕДЕНИЯ ОБ АВТОРАХ

СИБГАТУЛЛИНА Айгуль Ильдаровна – магистрант, Казанский (Приволжский) федеральный университет, г. Казань.

Aigul Ildarovna SIBGATULLINA – Master's student, Kazan (Volga region) Federal University, Kazan.

Email: aigul.sibgatulli@gmail.com
ORCID: 0000-0003-4014-9558

ЯКУПОВ Азат Шавкатович – старший преподаватель, Казанский (Приволжский) федеральный университет, г. Казань.

Azat Shavkatovich YAKUPOV – Senior Lecturer, Kazan (Volga region) Federal University, Kazan.

E-mail: azat.yakupov@it.kfu.ru
ORCID: 0000-0002-2333-8819

Материал поступил в редакцию 6 июня 2022 года