Обзор баз данных временных рядов

Светлана Гаврикова

Аннотация — В этой статье рассматриваются временные ряды как тип данных, их использование в Интернете вещей (IoT) и наиболее популярные платформы для хранения временных рядов (базы данных временных рядов). В настоящее время Интернет вещей остро нуждается в эффективных и оптимизированных базах данных временных рядов и системах управления для работы с разнородными данными. Системы управления базами данных (СУБД) временных рядов занимают второе место по популярности категорий на db-engines.com. Основой выборки баз данных для анализа стала в основном верхняя часть рейтинга СУБД db-engines.com. В статье рассмотрены структуры БД, возможности агрегирования данных, гибкость масштабирования, оптимизацию хранилищ, безопасность, возможность использования вспомогательных инструментов, синтаксис запросов, включая примеры с временными рядами, полученными с датчиков. Основная цель этой работы – изучить реальные платформы с их плюсами и минусами, подчеркнуть необходимость осознанного подхода к выбору инструмента, а также критическую важность поиска инновационных технологий для хранения временных рядов ради будущего всего цифрового общества.

Ключевые слова — Базы данных временных рядов, временные ряды, СУБД временных рядов, хранилища данных временных рядов, influxdata, clickhouse, mongodb, opentsdb, timescale, kdb, prometheus, graphite, amazon timestream

I. Введение

Рост популярности специализированных хранилищ, в особенности хранилищ временных рядов, которые смогли обойти по популярности разве что графовые базы, связан растущим уровнем интеграции информационных технологий в окружающую среду человека и необходимостью оптимизации плотно связанной со спецификой данных. Растет необходимость осуществлять интеграцию, контроль и анализ на разных уровнях и различных объемов данных, которые требуют специфических подходов к гетерогенным данным.

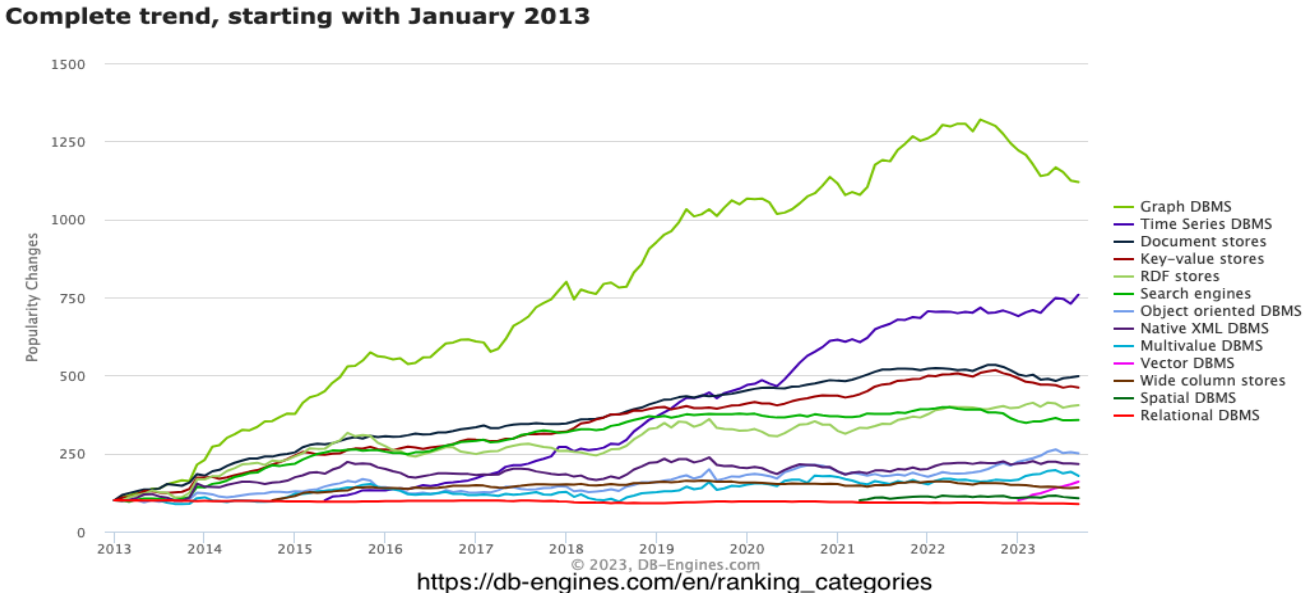

Рис.1 Тренд рейтингов специализированных СУБД, db-engines.com (2013-2023 гг.)[7]

В глаза бросается также факт, что СУБД графовых БД теряют популярность уже год, тогда как рейтинг СУБД временных рядов практически неуклонно и стабильно растет с 2013 года [7].

Большие данные (Big Data), львиную долю которых составляют временные ряды IoT, IIoT(промышленный IoT), Space IoT (космический IoT) и т.п., а также трафик социальных сетей и различных экосистем, — весь, так называемый, highload — требует огромных ресурсов и, следовательно, оптимизации их использования, поскольку физический ресурс памяти всегда ограничен и не может расти равно пропорционально масштабам данных. Вездесущие Big Data — термин, обозначающий технологию обработки структурированных и неструктурированных данных больших объемов для получения полезных человеку результатов [3].

II. Временные ряды. Сфера применения

Временные ряды (Time series) — это данные, претерпевающие некоторые изменения с течением времени, и фиксируемые в конкретные промежутки

времени. Временные ряды составляют разнородные метрики, фиксированные в определенный момент, за показателями которых закреплен временной отрезок. Простой пример графика с двумя временными рядами: фактическая температура воздуха и температура по ощущениям, как на рисунке 2.

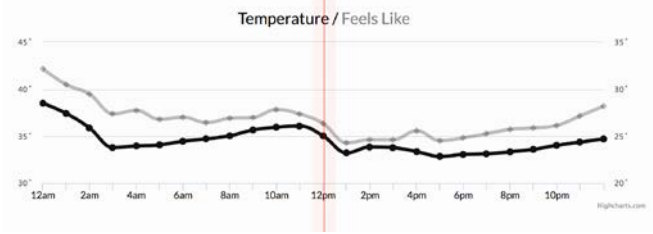

Рис.2 Пример визуализации на графике двух временных рядов: Температура(черным)/Ощущается как(серым), https://www.highcharts.com/[34]

Например, временными рядами считают электрические импульсы в мозгу, скачки цен на биржах, частоту запросов к серверу, метеорологические замеры, данные с датчиков об активности на предприятии и прочее. Самый наглядный и всем знакомый пример мониторинга временных рядов – сердечная электрокардиограмма (ЭКГ).

Говоря про термин, N-компонентным векторным временным рядом (N-мерным) называют совокупность функций X по n от t, где t – переменная соответствующая времени выполнения или регистрации наблюдений и измерений, принимающая значения 0, +- 1, +- 2 и т.д. [2]. Главным методом, используемым для анализа временных рядов, является гармонический анализ или фурье-анализ. Эксперименты ограничиваются инвариантными по отношению к временным сдвигам. Т.е. доля значений X(t), t > u в некотором интервале I должна быть примерно такой же, как доля значений X(t), t > u + v, попавших в интервал I для всех v.

Типичные физические эксперименты имеют свойство временной инвариантности. Поскольку люди извлекают уроки из прошлого и соответственно меняют свое поведение, ряды, относящиеся к человеческой деятельности, вообще говоря, не являются инвариантными по времени. В то же время, стоит отметить, что стационарную компоненту можно выделить и из нестационарных рядов.

Фильтры – важный класс операций над временными рядами – проще всего описываются и исследуются средствами гармонического анализа. Второе важное требование к временным рядам – они должны иметь короткий промежуток времени зависимости. С течением времени ряды подвергаются случайным возмущениям, не связанным с их предысторией, и эти случайные толчки в конечном счете формируют основное содержание процессов. Требование к временному ряду – обладать слабой «памятью» – условие перемешивания. Основным способом построения оценок в статистике временных рядов является метод моментов. Для обоснований широко привлекается асимптотическая теория. При исследовании временных рядов широко применяются два подхода: стохастический (вероятность и статистика) и функциональный [2].

III Базы данных. Особенности работы с временными рядами

Что же специфичного в работе с временными рядами в сравнении с любыми другими типами данных, что обуславливает необходимость особого подхода к разработке и обслуживанию базы данных? Для того, чтобы понять такие особенности, стоит перечислить где и как используют временные показатели с особым вниманием. Для интернета вещей временные ряды являются источником данных о работе датчиков/сенсоров в реальном времени, где для решения задач временные метки имеют критическую важность, как, например, в случае с датчиком прихода на работу сотрудников или датчиком сейсмической активности. Особые требования к хранению временных рядов обусловлены спецификой работы с ними. Во-первых, временные ряды всегда линейно нарастают. В отличие от базы данных, где для нас актуально изменять данные за любой период, базы хранения, оптимизированные для временных рядов, нуждаются в стабильном линейном наращивании объемов информации (масштабировании) и отсутствия необходимости изменять что-то в прошлом, сохраняя данные как «исторические». Метрики, которые уже зафиксированы в прошлом, являются зафиксированным фактом и манипулируем этими данными мы только на уровне их представления, что тоже важно и должно быть учтено в возможностях работы с базой данных. Значимой особенностью является необходимость работы с временными рядами за пределами показателей базы данных. Необходим быстрый и удобный механизм манипулирования временными рядами на основе внешних для базы метрик, составление сводных аналитических отчетов в комплексе с внешними данными. Отсюда, важное значение приобретают возможность и удобство интегрирования с популярными сервисами и приложениями по работе с данными, как облачными, так и локального хранения. Также важна оптимизация памяти под все эти цели. Сложно переоценить важность безопасности баз данных, ведь многие показатели, попавшие в руки недобросовестных лиц, могут повлечь серьезные проблемы для корпораций, государств и отдельных людей, а утеря данных с фиксированными показателями состояния природы могут стать причиной ошибок дальнейшего прогнозирования климатических катастроф и пр.

Есть несколько моделей баз данных: иерархические, сетевые, и реляционные. Последние являются самыми распространенные с момента их появления. Реляционная модель, разработанная в 70-ые гг Эдгаром Коддом, обладает двумя отличительными особенностями: обеспечивает представление в виде двумерных таблиц и над такими таблицами можно осуществлять алгебраические операции [3]. Классические реляционные хранилища обеспечивают наилучшее сочетание простоты, устойчивости, гибкости, производительности, масштабируемости и совместимости, однако их показатели по каждому из пунктов не обязательно выше, чем у аналоговых систем, ориентированных на какую-то одну особенность. Согласно Майклу Стоунбрейкеру,

пионеру исследований в области больших баз данных, идея «безразмерности», когда традиционная архитектура СУБД, изначально разработанная и оптимизированная для обработки бизнес-данных, используется для поддержки приложений, требующих обработки больших объемов данных, больше не применима к рынку баз данных. Мир коммерческих СУБД будет дробиться на набор независимых, специализированных средств управлениями базами данных. В последнее десятилетие (прим.: источник 2018 г. публикации) появилось множество различных задач, связанных с обработкой больших объемов данных, для которых реляционные СУБД обеспечивают плохое соотношение «цена/производительность» и при решении которых пришлось отказаться от использования этих СУБД. Все популярные реляционные СУБД, — MS SQL Server, DB2 или напрямую реализованный интерфейс System R в первом выпуске Oracle, — все они берут свое начало от System R, разработанной в 70-ые гг прошлого века. Современные СУБД, в большинстве своем, остаются архитектурно идентичными System R, хотя сегодня процессоры и обладают тысячекратно большей мощностью, а память — тысячекратно большей емкостью. Однако, стоит отметить, что пропускная способность между диском и основной памятью растет гораздо медленнее. Что действительно сильно изменилось со времен написания первой реляционной базы, — это рынок. Сегодня рынок данных выходит далеко за пределы некогда актуальной и сильно ограниченной работы с «бизнес-данными» [3]. Специфика реляционных БД не оптимальна для временных рядов по нескольким причинам. Устройства в системе (IoT), чаще всего, асинхронны, работают по собственным циклам, со своими временными интервалами. Это означает, что каждая строка в таблице будет содержать только часть из потенциально возможных атрибутов. Остальные атрибуты будут пустыми (null value). Это ведет к неэффективному расходованию дискового пространства и затрудняет последующую обработку. Вторая причина в «урезанном» функционале необходимом для работы с временными рядами, по сути, представляющем из себя последовательное чтение данных [5]. Реляционные СУБД предполагают два основных класса приложений: обработка транзакций в реальном времени (OLTP, Online Transaction Processing)[29] и интерактивная аналитическая обработка данных (OLAP, Online Analytical Processing) [30]. В приложениях OLTP под транзакцией понимают набор последовательных операций модификации информации в базе данных, которые рассматриваются как неделимая единица работы с данными и переводят базу данных из одного согласованного состояния в другое. Результатом транзакции СУБД является фиксация (успешное выполнение) или откат (неудачное выполнение) всех входящих в нее операций. Типичные сценарии использования временных рядов, показывают, что СУБД-ВР не нуждаются в поддержке OLTP: данные накапливаются и могут архивироваться для экономии объема дисковой памяти, но операции модификации и удаления данных отсутствуют. Однако в случае с СУБД-ВР обработка данных по сценарию OLAP востребована в урезанном варианте: в этом случае время является фактически единственным измерением, агрегация по которому не требует OLAP-куба, поскольку представляет собой примитивную операцию (например, нахождение минимального или максимального значения, стандартного отклонения временного ряда и др.).

Поскольку человечество вступило в век BigData, — хранящиеся на мощнейших серверах разнородные данные, поступают на огромных скоростях в data-центры по всему миру и измеряются в петабайтах (миллионах гигабайтов), задачи сжатия данных нельзя переоценить. Например, Нью-Йоркская фондовая биржа проводит 4,2 миллиардов транзакций в день, 162 тысячи транзакций в секунду и суммарно 5,4 триллиона транзакций за 5 лет, данные о которых хранятся в системе [13].

Основные поставщики СУБД реализуют системы хранения, ориентированные на запись, где атрибуты каждой записи располагаются в области хранения непрерывно – строчные СУБД. При использовании этой архитектуры для выталкивания на диск всех атрибутов одной строки требуется вывод на диск блока таких строк. Иными словами, это системы, ориентированные «на запись». В таких системах легко достигается высокая производительность при выводе строк. Основная рабочая нагрузка систем хранилищ данных состоит из непредвиденных запросов, затрагивающих большие объемы «исторических» данных. В связи с этим они нуждаются в оптимизации в расчете «на чтение». Существенно более эффективной для таких задач оказывается модель хранения «по столбцам», предполагающая непрерывное хранение значений одного атрибута во всех строках – колоночные СУБД. В общем случае – это разделение таблиц на столбцы, которые размещаются в физической памяти раздельно друг от друга. При таком способе хранения данных считываются только те атрибуты, которые нужны для обработки запроса и в основную память не переносятся лишние данные. У строк появляется все больше атрибутов (сотни) и такой подход существенно улучшает производительность для хранилищ данных, где типичные запросы включают агрегаты, вычисляемые на небольшом числе атрибутов крупных наборов данных.

Сжатие данных играет одну из ключевых ролей в работе с временными рядами ввиду больших объемов данных, поступающих в реальном времени и накапливающихся. Колоночное хранение позволяет улучшить показатель сжатия по сравнению со строчными системами. Некоторые из таких алгоритмов: RLE (кодирование длин серий), словарный метод (использование пар код и данные или слова, при сжатии слова заменяются на их коды), векторное кодирование(значение сопоставляется с битовой строкой), алгоритм Лемпеля-Зива-Велча(динамическое создание таблиц преобразования строк: последовательность ставится в соответствие группе бит) и др. [3].

Базы NoSql в данном сравнении немного выигрывают. Чтобы решить накопившиеся проблемы реляционных баз данных и появилась альтернатива в виде баз NoSQL (что означает «не только SQL»).

NoSQL используется Amazon, Google, Facebook, Twitter, eBay и др. для реализации собственных хранилищ с целью хранения больших объемов данных. NoSQL базы имеют ряд преимуществ. Они поддерживают горизонтальную масштабируемость, что особенно актуально, когда объем данных растет с огромной скоростью, разделение данных и их дублирование(репликацию), что обеспечивает скорость и высокую отказоустойчивость системы, данные хранятся в виде <ключ, значение> и не требуют разработки схемы, агрегат в поле «значение» в свою очередь может иметь сложную структуру (например, Json-документ), инсталляция в кластер такой базы значительно проще развертывания параллельной реляционной базы данных, кластер база NoSQL может быть реализован на большом числе маломощных узлов и помещен в облако, многие NoSQL поддерживают параллельные вычисления. Минусы NoSQL в невозможности использовать там, где важны ACID-транзакции, задержка во времени репликации может повлечь ошибку прочтения устаревших данных и необходимость использования процедурного языка программирования, накладывая дополнительные ограничения на возможности работы аналитиков [3]. Они более гибкие, но количество источников информации сильно ограничено. Реляционные СУБД не оптимизированы для хранения временных рядов. Их механизм оптимизирован под постоянное обращение к данным и манипуляции с данными. А значит с большими объемами данных, которыми являются временные ряды, которые не нуждаются в таких манипуляциях, механизмы будут работать медленнее. Чтобы получить максимум от вашей платформы работы с IoT, база данных должна отвечать следующим требованиям: поддерживать разнообразие входящих данных, изменчивость их размера и наполнения, гибкую масштабируемость и оптимизированность для работы с временными рядами [28].

Одной из самых актуальных сфер использования баз, оптимизированных для хранения временных рядов, — среда IoT. Здесь огромные потоки данных в реальном времени, получаемые с разных устройств, нужно эффективно собирать и хранить максимально компактно, так как ресурс физической памяти ограничен и очень дорог.

II. Платформы и сервисы для работы с хранилищами временных рядов

Актуальный рейтинг баз данных временных рядов от db-engines.com на сентябрь 2023 года (прим.: скриншот сделан 23.09.23) показан на рисунке 3.

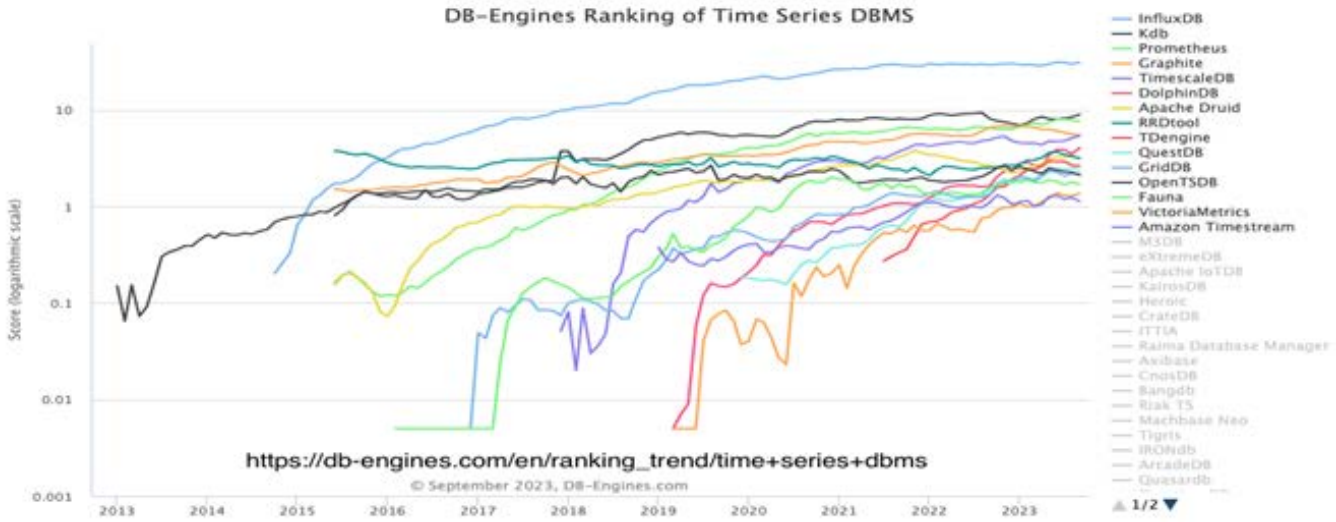

Рис.3 Рейтинг СУБД временных рядов db-engines.com (2013-2023)[7]

Топ-15 СУБД ресурса db-engines.com выглядит следующим образом:

1. *InfluxDB*
2. *Kdb+*
3. *Prometheus*
4. *Graphite*
5. *Timescale DB*
6. DolphinDB
7. Apache Druid
8. RRDtool
9. TDengine
10. QuestDB
11. GridDB
12. *OpenTSDB*
13. Fauna
14. Victoria Metrics
15. *Amazon Timestream*

… etc.

Сервисы, выделенные в рейтинге курсивом, мы рассмотрим в рамках этого материала, а также неучтенные в этом рейтинге сервис Яндекса *Click House*, завоевавший популярность и за пределами России, и известную платформу *MongoDB.*

A. InfluxDB (InfluxData)

Influxdata предлагают управление всеми типами данных временных рядов в единой базе, гибкую масштабируемость. Работа с БД возможна в облаке или локально, или в комплексе. Influxdata позиционируют себя как БД, созданная и оптимизированная непосредственно для работы с временными рядами. Данные организуются в соответствии с линейным протоколом. Протокол основан на четырех компонентах: измерение

(эквивалентно таблице в базе данных), метка (сет меток), область (или сет областей) и метка времени с точностью до наносекунд (одна миллиардная доля секунды). Каждая уникальная комбинация измерения и сета меток представляет серию данных. Чем больше серий, тем больше мощность базы. Оборудование пользователя определяет порог мощности, так как ключи серий хранятся в виде индексов на диске.

Первоначально пользователь устанавливает на компьютер интерфейс командной строки (CLI) Influx и создает профиль подключения к облаку InfluxDB посредством введения идентификационные номера аккаунта и кластера. В качестве примера создадим базу данных my_db. Также при создании базы мы вводим срок хранения, укажем для примера 3 года:

```
influxctl database create --retention-
period 3y my_db
```

Следующим шагом создают токен с правами. Токен рекомендуется хранить в безопасном месте. Предоставим права на чтение и запись данных к нашей базе my_db:

```
influxctl token create \
--read-database my_db
--write-database my_db
```

Для записи данных InfluxDB использует линейный протокол, понимание которого не является необходимостью, если использовать такие инструменты записи и получения данных, как серверный агент сбора данных Telegraf и InfluxDB клиентские библиотеки, которые работают с линейным протоколом вместо пользователя. Но можно и работать напрямую с линейным протоколом посредством командной строки терминала или InfluxDB HTTP API (v1 и v2). Синтаксис линейного протокола представляет из себя комплекс из названия измерения (measurement), тэг ключ-значение по желанию, необходимый ключ-значение поля и временная метка в наносекундах, если иное не обозначено, которая не является необходимой. Первый пробел отделяет пары ключ-значение поля от пары ключ-значение тэга, второй пробел отделяет временную метку:

```
// Синтаксис
<measurement>[,<tag_key>=<tag_value>[,<
tag_key>=<tag_value>]]
<field_key>=<field_value>[,<field_key>=
<field_value>] [<timestamp>]
// Пример
my_measurement,tag1=value1,tag2=value2
fieldKey="fieldValue"
1556813561098000000
```

Формат данных предполагает возможность использования беззнаковых целочисленных значений, временных меток в формате Unix, также линейный протокол поддерживает использование спец. символов, включая эмодзи. Пример кода:

```
// Имя измерения с пробелом
my\ Measurement fieldKey="string value"
// Двойные кавычки
myMeasurement      fieldKey="\"string\"
within a string"
// Ключи и значения тэгов с пробелами
myMeasurement,tag\            Key1=tag\
Value1,tag\      Key2=tag\      Value2
fieldKey=100
```

Пример данных линейного протокола домашних сенсоров:

```
flat,room=kitchen
temp=22.2,hum=43.3,co=0i 1691827200
flat,room=children
temp=23.1,hum=45.0,co=0i 1691827200
flat,room=Master\               bedroom
temp=21.1,hum=44.1,co=1i 1691830800
flat,room=children
temp=23.0,hum=45.0,co=0i 1691827200
```

В данном примере с датчиков собирают следующие данные: temp – температура, hum – влажность и co – содержание оксида углерода в воздухе.

InfluxDB поддерживает работу с традиционными SQL-запросами или использование схожего с SQL языка InfluxQL, созданного для написания запросов к базе временных рядов Influx. Язык использует несколько операторов, включая `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `AS`, `AND`, `OR`. Например, запрос может выглядеть так:

```
SELECT temp AS "temperature"
FROM flat
WHERE time >= '2022-07-03T14:05:00Z'
```

В результате мы получим таблицу с данными по температуре, начиная с указанной даты и времени до самых последних полученных показателей. Или, например, такой запрос:

```
SELECT *
FROM flat
WHERE room = 'kitchen'
```

Результатом этого запроса будет выдача данных всех имеющихся в базе
«дом» показателей датчиков на кухне.

Платформа Influx сотрудничает с CISCO, IBM, PayPal в процессах выстраивания работы с IoT в реальном времени, аналитикой и облачными данными временных рядов. InfluxDB способна принять миллионы входящих значений в секунду и быть интегрированной с сотнями систем и решений [6].

Б. KDB+(KX Systems)

KX Systems предлагают KDB+ — облачную мульти вертикальную платформу для анализа потоков. KDB+ — это колоночная БД, которую позиционируют, как самую быструю в мире базу данных временных рядов и механизм для аналитики. Доступен облачный вариант базы на разных языках. Высокая скорость запросов достигается за счет вычислительного

механизма, находящегося в памяти. База имеет гибкую масштабируемость и использует q — язык программирования, используемый как язык запросов для анализа массивов, на который KX Systems имеют уникальные права. Язык q сериализует и сохраняет объекты в виде единого двоичного файла. Таблицы представлены в файловой системе в виде одноименных каталогов и двоичных файлов. Сериализация данных зависит от размера и как предполагается использовать таблицу. Сериализация объекта создает единичный бинарный файл, — лучший вариант для малого размера и если в запросах будет участвовать большинство колонок, splayed table — каталог файлов, оптимален для размера до 100 млн записей, partitioned table — таблица разделенная по определенному показателю, например, по дате, с развернутой таблицей для каждой даты, и segmented database — секционированная таблица, распределенная по дискам. Разделение данных на несколько устройств хранения позволит хранить очень большие данные, каковыми являются, как правило, данные IoT, распараллеливать запросы и оптимизировать обновления [8].

Язык q имеет свой особенный синтаксис qSQL, который включает 170 ключевых слов, санитарные и зарезервированные слова. Ключевые слова составляют следующие категории: контроль, среда, интерпретация, ввод/вывод, итерировать, объединить, лист, логика, математика, мета, запрос, сортировать, таблица и текст. Язык предполагает возможность использования таких операторов, как стандартные: сложение, вычитание, умножение и остаток от деления (+, -, *, %), сравнение: равные (=) для числовых переменных, а также форматов лист, словарь и таблица, неравные (<>) и сравнение (~) с возвращение логической переменной, независимо от типа сравниваемых переменных, также логические больше/OR(|) и меньше/AND(&), оператор замены(^) и оператор '#', позволяющий забрать некоторые элементы из листа, словаря, таблицы. В q отдельно проработаны и оптимизированы операции совмещения списков с разделением элементов некоторым разделителем. Используют операторы 0: – для текстовых файлов, 1: – для бинарных файлов, 2: – для динамической загрузки. Например, выражение X 1: Y или 1:[X;Y]. Здесь на вход мы подаем вложенный список X, содержащий два элемента – список символов, по одному на поле, и список целочисленных значений ширины, также по одному на поле. Сумма ширины полей во втором списке будет равняться ширине записи. Y – файловый дескриптор, строки или последовательность байтов, матрицу которого мы получим на выходе. То есть на выходе мы получаем список списков, где каждому полю соответствует один список. Также очень удобно с помощью языка q можно легко составить таблицу из полученных результатов датчика или сенсора, используя символ-разделитель для определения ячеек. Например, команда вида `(Y ; ",") 0: 'file_name'` , где Y тот же список списков, создаст таблицу на основе данных текстового файла file_name, разделив данные, отделенные запятой, по строкам, названия столбцов будут взяты из первого показателя строки файла. Аналогично схема работает для бинарных файлов `Y ; ",") 1:binary_file_name'`. Особое внимание в q уделили эффективности и удобству межпроцессному взаимодействию на одной машине, в рамках одной сети или разных сетей, в интернете, там, где мы можем получить доступ. Коммуникация может быть синхронизированной, с ожиданием возвращаемого результата, или асинхронной (без ожидания и без возвращаемого значения). Дескриптор соединения имеет вид: `': [server] : port` , где в квадратных скобках опционально указывают идентификатор сервера, а после двоеточия добавляется номер порта. В рамках сети мы можем обращаться к машине по имени, также использовать IP-адрес или URL [23].

Первое значительное отличие qSQL от традиционного SQL в упорядоченности столбцов и колонок, что особенно удобно для временных рядов, которые при добавлении в таблицу сохраняют свой хронологический порядок. Второе отличие состоит в том, что физически таблицы q хранятся, как коллекции списков колонок, что делает векторными операции с данными, а манипуляции с простым одноколоночным списком, в формате атомарных, агрегатных функций и унификации, и вовсе просты и особенно быстрые, так как сводятся к прямой адресации в памяти. qSQL поддерживает upsert семантику, что представляет собой процесс обновления (UPdate + inSERT) значения существующих строк словарей, если по данному ключу в таблице уже существует значение. Но такая процедура неинтересна в контексте работы с временными рядами, так как мы добавляем значения с временной меткой и не изменяем их в будущем. Синтаксис добавления записи в таблицу выглядит так:

```
q)table_name,:`name`iq!(`Dent; 98.0)
'type
Например: q)table_name,:(`Prefect; 126)

//Добавление нескольких записей
q)table_name:([]
name:`Dent`Beeblebrox`Prefect; iq:42 98
126)
```

Также для этих целей работает SQL-оператор `INSERT`:

```
q)`table_name                    INSERT
(`name`iq)!(`Slartibartfast; 134)
,3

//Синтаксис запроса к БД на qSQL:
SELECT <p(s)> <by p(b)>
FROM t(exp) <where p(w)>
```

`SELECT` и `FROM` – обязательные ключевые слова, t(exp) – выражение таблицы (table expression) – любое выражение, результатом которого будет выдача таблицы или таблицы ключей, p(s) – необязательное `SELECT`, p(b) – необязательное BY, p (w) –

необязательное WHERE.
В отличии от SQL в q-SQL каждый столбец фразы BY автоматически включается в ключевые столбцы результата без дублирования в выражении SELECT. Иными словами, мы группируем по вычисляемым столбцам. В примере ниже мы запрашиваем усредненные значения неких абстрактных домашних сенсоров с временными рядами в интервале 100-миллисекунд:

```
q)table_name:([]Time:00:00:00.000+til
1000000;Room:1000000?`Time`Room;Sensor:
1000000?100.)
q)SELECT AVG Sensor BY 100 xbar Time,
Room FROM table_name
Time|            Room| Sensor
--------------| --------
00:00:00.000 Kitchen | 55.26494
00:00:00.000 Living_room | 41.81758
00:00:00.100 Kitchen | 48.88826
00:00:00.100 Living_room | 46.10946
```

Удобно применение оператора BY с пустым выражением SELECT, что даст нам результирующий набор эквивалентный результату запроса LAST ко всем колонкам:

```
q)table_name:([]     desk:`a`b`a`b`a`b;
acct:`1`2`3`4`1`4; pnl:1.1 -2.2 3.3 4.4
5.5 -.5)
 q)SELECT BY desk FROM table_name
 desk| acct pnl
 ----| ---------
 a   | 1    5.5
 b   | 4    -0.5
```

Несколько примеров запросов к таблице:

```
//Отсортировать значения по возрастанию
с помощью XASC, в убывающем – XDESC
q)table_name:([] c1:`a`b`c`a; c2:20 10
40 30)
q)`c2 XASC table_name
c1 c2
-----
b  10
a  20
a  30
c  40
q)`c1`c2 XASC table_name
c1 c2
-----
a  20
a  30
b  10
c  40

//Запрос уникальных значений
q)SELECT     distinct     FROM     ([]
Room:`Kitchen`Living_room`Kitchen;
Temperature:22.1 23.3 22.1)
```

Результирующая таблица примет вид:

```
Room Temperature
-----
Kitchen  22.1
Living_room  23.3

//Запрос данных столбца c1 и
нумерование этих значений в первом
столбце с //помощью ключевого символа
i, неизвестные названия столбцов
выводятся как X
q)select i, Room FROM table_name
```

Результирующая таблица примет вид:

```
X Room
----
0 Kitchen
1 Living_room
2 Childrens_room

//Получить значения в заданном
диапазоне
q)table_name:([]
Room:`Kitchen`Living_room`Childrens_roo
m; Temperature:21.1 22.0 23.1; Co:0.9
0.5 0.4)
q)table_name                          WHERE
table_name[`Temperature]>21.5
Room Temperature Co
---------
Living_room  22.0 0.5
Childrens_room  23.1 0.4
q)SELECT    FROM    table_name    WHERE
Temperature>21.5
_

//Суммирование и среднее значение с
помощью встроенных агрегатных функций
SUM и AVG
q)SELECT  total:SUM  qty,  mean:AVG  qty
from sp
total mean
--------------
3100  258.3333
```

Язык q дает возможность создавать связи между таблицами, как статически, так и динамически, что сильно отличает его от SQL, где связи таблиц реляционной БД устанавливаются статически на языке DDL. Таблицы в q являются сущностями (объектами) первого класса. В q-SQL возможно динамически создавать связи между таблицами, которые могли бы их иметь, но статически они не были определены.
С помощью q было реализовано упрощение процесса работа с большими данными – проблема выбора изучения языка, чтобы размещать функции непосредственно рядом с данными, либо выгрузка данных для последующего выполнения вычислений, здесь отсутствует. q-SQL является языком хранимых процедур и в то же время обладает достаточной мощностью для обработки больших данных. Параметры функции могут использоваться для предоставления определенных значений для запросов.

В частности, шаблоны `SELECT, EXEC, UPDATE, DELETE` могут быть вызваны внутри функции с параметрами для получения параметризированного запроса.

Мы рассмотрели далеко не все особенности языка, в том числе, особенности объединения таблиц. Можно сделать вывод, что синтаксически q достаточно сильно схож с традиционным SQL, и в то же время обладает рядом преимуществ для работы с временными рядами.

B. PROMETHEUS

База данных временных рядов Prometheus входит в систему мониторинга и оповещений Prometheus.

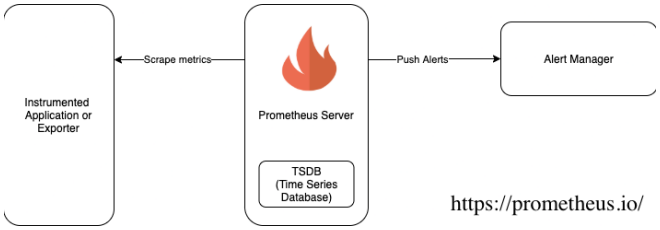

Рис.4 Структура системы мониторинга Prometheus[10]

База является локальной, но также опционально интегрируется с удаленными системами хранения. Локальные хранилища имеют ограничения, ввиду того, что они не кластеризуются и не реплицируются, и, как следствие, не являются гибко масштабируемыми, долговечными и устойчивыми к сбоям. В частности, Non-POSIX системы не подходят для работы с Prometheus, такие как большинство реализаций NFS ввиду риска неисправимых повреждений. Разработчики рекомендуют использовать для резервного копирования моментальные снимки, а для обеспечения доступности – RAID (Redundant Array of Independent Disks) – избыточный массив независимых/недорогих дисководов объединяет два или более дисководов с целью создания системы дисков с новыми свойствами: максимально возможный объем с увеличением скорости и надежности работы и сохранение данных в случае отказа части оборудования [22]). Базовыми составляющими Prometheus являются три элемента: сервер, целевые объекты (например, из приложения, предоставляющего свои показатели, которые подлежат дальнейшему удалению) и диспетчер оповещений на основе заданных правил. Чтобы отслеживать нужные нам метрики, нужно установить инструментарий путем добавления кода из клиентской библиотеки Prometheus и указать нужные метрики. Далее нужно настроить, чтобы Prometheus собирал информацию о состоянии метрики в определенном режиме, например, с интервалом в 1 минуту. Данные суммируются, и выводится суммарная метрика к данной минуте, сохраняясь и создавая собственно временной ряд (рис.5).

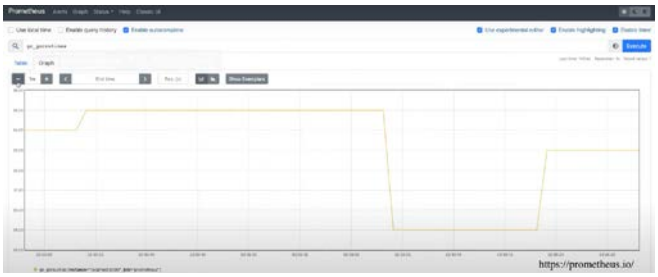

Рис.5 Пример представления данных временных рядов в Prometheus[10]

У Prometheus есть API, которое позволяет запрашивать метрики, которые были собраны, создавать на основе метрик диаграммы, дэшборды. Синтаксис PromQL используют для запросов к этим данным. При хранении на диске, полученные данные группируются в блоки по 2 часа, каждый блок состоит из каталога, содержащего подкаталог chunks, содержащий все выборки временных рядов за этот промежуток времени, файлы метаданных и файл с индексами(индексирует имена показателей и метки каталога chunks). Образцы каталога по умолчанию сгруппированы в один или несколько сегментных файлов до 512 Мб каждый. При удалении через API, удаленные записи сохраняются в отдельных файлах tombstone вместо полного немедленного удаления, что помогает избежать критических ошибочных процедур. Также для сохранности данных у системы есть журнал предварительной записи необработанных данных. Данные хранятся на локальных дисках, либо во внешних хранилищах, которые интегрируются с системой. Масштабируемость и долговечность хранения данных локального хранилища временных рядов Prometheus ограничена, вместо этого предлагается набор интерфейсов для интеграции с другими удаленными системами хранения [10].

Пример запроса на PromQL, который возвращает временные ряды с метрикой «Температура» и имеющие метки «Комната = Кухня» и «Дом=Квартира50»:

```
temperature{room="kitchen",flat="flat50
"}
```

А такой запрос соберет временные ряды температуры на кухне, в гостинной, детской, но исключая данные квартиры 103:

```
temperature{room=~"kitchen|living_room|
childrens_room",flat!="flat103"}
```

Временной интервал можно задать в секундах, минутах, часах, днях, неделях и годах. Путем конкатенации можно объединить временные интервалы, блоки должны быть запрошены от длинного к короткому, например:

```
5h 1h30m 5m 10s
```

Чтобы получить данные температуры назад во времени за определенный временной диапазон, например, за 12 минут включительно, нужно написать подобный запрос:

```
temperature{room="Kitchen"}[12m]
```

Можно задать смещением во времени. Такой запрос вернет 12-минутный показатель, который был неделю назад:

```
rate(temperature [12m] offset 1w)
```

В PromQL работают функции `absent(v instant-vector)` и `absent_over_time(v range-vector)`, которые возвращают «пустые» вектора, функция `changes(v range-vector)` возвращает количество раз, когда значение ряда менялось за период времени, `delta(v range-vector)` считает разницу между первым и последним значением каждого значения временного ряда в диапазоне вектора v, `deriv(v range-vector)` вычисляет посекундную производную ряда в диапазоне вектора v, `holt_winters()` для работы с датчиками, которая выдает сглаженные значения, `irate()` вычисляет посекундную скорость увеличения элементов ряда, `predict_linear(v range-vector, t scalar)` предсказывает значение временного ряда на t секунд вперед, используя простую линейную регрессию, и др. полезные функции. Есть также тригонометрические функции и экспериментальные функции для работы с гистограммами – `histogram_count()`, `histogram_sum()`, `histogram_fraction()`, `histogram_fraction()` и др.

Prometheus предпочли для работы в крупном российском маркетплейсе LaModa, отмечая хорошую работу с большим объемом входящих данных, на их опыте лучший, чем опыт теста на Influxdata. Кодирование на уровне каталога chunk в Prometheus было реализовано в объеме 500 тыс метрик каждые 15 секунд в течение 50 дней, объем реальной памяти составил 2,502 Тб, а реальной физической памяти было занято 493 Гб. В сборе данных есть один нюанс: если данные обновляются непостоянно и chunk не обновлялся уже 1 час, то каталог закрывается и сохраняется, что увеличивает объем занимаемой памяти при нерегулярном поступлении данных [19].

В сравнении с типами данных, которые поддерживает Influx, Prometheus обладают меньшим разнообразием. Против наносекунд, float64, int64, bool и string у Influx, в инструментарии Prometheus float64, ограниченная поддержка string и временные метки с расширением в миллисекундах. Influx используют разновидность LSM-дерева с журналом предварительной записи для хранения, разделенный по времени, что, по признанию разработчиков Прометея, гораздо больше подходит для ведения журнала событий, нежели подход Прометей, основанный на добавлении файла для каждого временного ряда. На сайте Прометея рекомендуют выбрать Influx, если вы ведете журнал событий, нуждаетесь в согласованном представлении данных между репликами и ищете базу для долгосрочного хранения данных, такой сможет обеспечить коммерческий пакет Influx с кластеризацией. Прометей же рекомендуют ищущим продукт для работы с метриками, хотите инструмент высокой степенью доступности и безотказной работы для построения графиков и получения оповещений, уведомлений. Здесь же стоит отметить: Prometheus – продукт с открытым исходным кодом и, казалось бы, Influx тоже, но коммерческий пакет, включающий кластеризацию, все же имеет закрытый код [25].

Г. Graphite и The Whisper DB

Инструмент мониторинга метрик Graphite является одним из популярных сервисов работы с временными рядами. Как заявлено на ресурсе системы, «работает как на дешевом оборудовании, так и в облачной инфраструктуре». Graphite предлагает простой способ добавить измерения в базу временных рядов. Инструмент состоит из 3 программных компонентов: carbon – серверная часть хранилища, один или несколько демонов, которые прослушивают данные временных рядов, whisper – простая библиотека баз данных временных рядов, graphite webapp – веб-приложение Django, которое отображает графики по запросу с помощь Cairo. Изначально данные отправляются в Carbon, веб-приложение тут же получает к ним доступ и предоставляет пользователю возможности построения графиков, также предоставляя несколько способов создания и отображения, включая URL-API для рендеринга, позволяющего встроить график в веб-страницу. Graphite используют, например, с The Whisper Database — базой данных, схожей по дизайну и целям с RRD (round-robin-database, база данных с инструментами для работы с кольцевой базой), с фиксированным размером и имеющей специальную функцию, позволяющую преобразовать данные высокого разрешения (секунды на точку) в более низкое разрешение для оптимизации хранения исторических данных. The Whisper достаточно быстр для большинства задач, но медленнее, чем RRDtool (прим.: занимающая почетное восьмое место рассматриваемого нами рейтинга СУБД ВР, но не рассмотрена в рамках данной статьи). Данные хранятся на диске в виде чисел с плавающей запятой двойной точности. Временные метки существуют в формате UNIX. При добавлении записи в базу с несколькими архивами, данные записываются во все архивы [23].

Большинство функций в Graphite применяются к листу рядов, который может содержать один или несколько рядов, также есть функции принимающие на вход *seriesLists, которые принимают произвольное количество рядов. Например, можно применить функции агрегирования, такие как `average/avg`, `avg_zero`, `sum/total`, `count`, `range/rangeOf` и другие к серии:

```
//Синтаксис
aggregate(seriesList, func, xFilesFactor=None)
//Пример
&target=aggregate(host.cpu-[0-7].cpu-{user,system}.value, 'sum')
```

Функция `aggregateLine` принимает на вход метрику или лист рядов и рисует горизонтальную

линию, основанную на функции, примененной к ряду/рядам [27]:

```
//Синтаксис
aggregateLine(seriesList,
func='average', keepStep=False)
//Пример
&target=aggregateLine(server01.connecti
ons.total, 'avg')
```

Среди функций есть опция удаления пустых значений(None):

```
//Синтаксис
compressPeriodicGaps(seriesList)
//Пример
&target=compressPeriodicGaps(Server.ins
tance01.threads.busy)
```

Функция `delay(seriesList, steps)` позволяет сдвинуть все ряды на определенное число шагов, что удобно использовать, например, для вычисления производных:

```
//Пример
&target=divideSeries(server.FreeSpace,d
elay(server.FreeSpace,1))
```

Функция `group(*seriesLists)` берет произвольное количество рядов и объединяют их в один. Это полезно для функций, которые принимают на вход один.

Выделим функцию, позволяющую оценить количество попаданий в списки временных рядов, или обращений, за период времени. Функция работает с разными временными масштабами, что позволяет получить график для детализированных и списков с «сырыми» данными и изящно обработать редко поступающие события. Например, если передача данных не является постоянной, как в случае сбора показателей температуры в помещении, а подает данные в случае, например, срабатывания датчика дыма.

В Graphite есть ряд функций по работе с алгоритмом Холта-Винтерса для тренд-сезонных временных рядов, которые легко реализуются численно [24]. К данным можно применить функцию `holtWintersAberration(seriesList, delta=3, bootstrapInterval='7d', seasonality='1d')`, выполняющую прогноз на основе входящего ряда, которая строит график положительного или отрицательного отклонения данных от прогноза, схожую функцию `holtWintersConfidenceArea(seriesList, delta=3, bootstrapInterval='7d', seasonality='1d')`, которая строит область между верхней и нижней полосами прогнозируемых отклонений, функцию `holtWintersConfidenceBands(seriesList, delta=3, bootstrapInterval='7d', seasonality='1d')`, которая строит верхние и нижние полосы с прогнозируемыми отклонениями и функцию `holtWintersForecast(seriesList, bootstrapInterval='7d', seasonality='1d')`, где данные из начальной загрузки (по умолчанию одна неделя), предшествующего ряда, используются для начальной загрузки первоначального прогноза.

Альтернативой для работы с Graphite являлась The Ceres Database. В отличие от The Whisper размер базы Ceres не фиксирован и поддерживал разреженные данные произвольного разрешения. Т.е. можно было распределять временные ряды по нескольким серверам. По информации сайта Graphite, разработка базы приостановлена [17].

В целом Graphite является относительно пассивной базой данных временных рядов, функционал которой ограничен работой с языком запросов и функциями построения графиков. Остальные задачи решаются за счет подключения внешних инструментов [25].

Д. TIMESCALE DB

Timescale — облачная платформа/экосистема с возможностями фиксирования миллионов единиц данных в секунду, работающая на PostgreSQL — распространенной мощной объектно-реляционной системе баз данных с открытым исходным кодом с более чем 35-летним опытом активной разработки. Почему же реляционная база может эффективно работать с временными рядами? Сервис может похвастаться множеством расширений, дополнений и приложений для любой аналитики, это инструмент, зарекомендовавший себя на рынке, работающая на «чистом» SQL. Площадка заявляет миллисекундное время отклика на сложные агрегированные запросы к временным рядам. Предлагается выполнять сложный анализ данных быстро благодаря более чем 100 встроенным гиперфункциям SQL, таким образом сокращая код. Сервис предлагает алгоритмы сжатия, которые позволяют экономить более 90% места на диске. Благодаря политикам хранения данных и непрерывному агрегированию сервис автоматически сокращает объемы хранилища. Исторические данные предлагается оставлять в объектном хранилище, построенном на Amazon S3 (Amazon Simple Storage Service — сервис хранения объектов для хранилищ любой величины, обладающими всеми средствами защиты сервисов Amazon) и в целях операционного менеджмента сервис использует облачную инфраструктуру AWS.

PostgreSQL является одним из самых популярных инструментов для работы с временными рядами. По оценке Timescale позволяет ускорить разработку за счет увеличения скорости обработки на 44%, ускорения запросов в 350 раз и экономии места на 95%. К плюсам сервиса также относят возможность объединять реляционные таблицы и временные ряды. Для данных предоставляется до 16 Тб дискового пространства [11].

Набор агрегатных функций позволяет объединять строки, создавать массивы, функции принимают на вход широкий спектр форматов данных, включая пользовательский тип, и оптимизированы для больших потоков данных, что позволяет легко масштабировать базу данных.

Пример запроса для получения средней температуры во всем доме за весь период:

```
SELECT AVG(temperature) FROM building;
```

Пример запроса, который создает массив из всех показаний температуры по дому за весь период. Важно знать, что функция вернет null, а не пустой массив, если данных нет. Для исключения такой ситуации можно использовать функцию `COALESCE(value [, ...])`:

```
SELECT ARRAY_AGG(temperature) FROM building;
```

Функция ниже объединит все значения строк (пример: номера квартир дома) в произвольном порядке и установит разделитель между ними. В данном примере запятую:

```
SELECT STRING_AGG(flat ', ') FROM building;
```

Функции `MAX/MIN` вернут максимальное/минимальное значение показателя уровня углерода в здании:

```
SELECT MAX(co) FROM building;
```

Функция `AGE(timestamp1, [timestamp2])` вычисляет интервал между //двух временных меток. Если на вход подается только одна функция, рассчитывается интервал между текущим временем и заданным, как в примере:

```
SELECT AGE(TIMESTAMP '2020-01-01');
```

Функция `EXTRACT(field FROM source)` извлекает подполя временной метки (год, месяц, день и т.д.):

```
SELECT EXTRACT(YEAR FROM TIMESTAMP '2020-12-31');[29][30]
```

Также сработает `DATE_PART(field, source)`. Могут быть полезны гиперфункции – серия функций во временных рамках на традиционном SQL. Гиперфункции используют для вычисления процентильных аппроксимаций данных, вычисления средневзвешенных по времени значений, уменьшения выборки, сглаживания данных, а также для выполнения более быстрого подсчета отдельных значений с использованием аппроксимаций. Например, гиперфункции `hyperloglog()`, являющейся первым шагом для оценки количества различных значений в сырых данных, который поможет аппроксимировать количество уникальных значений:

```
hyperloglog(
    buckets INTEGER,
    VALUE AnyElement
) RETURNS Hyperloglog
```

Функцию `approx_count_distinct()` стоит использовать, если вы не уверены в количестве сегментов(buckets):

```
approx_count_distinct(
    VALUE AnyElement
) RETURNS Hyperloglog
```

После полученный hyperloglog(гиперлоглог) может использоваться для других гиперфункций.
Функция `rollup()` поможет объединить несколько промежуточных агрегатов гиперлоглог в единый промежуточный гиперлоглог. Например, можно объединить 15-минутные гиперлоглоги в дневной или дневные в месячный и т.п.:

```
rollup(
  hyperloglog Hyperloglog
) RETURNS Hyperloglog
```

Например, объединим данные гиперлогов двух числовых рядов: от 1 до 100 000 и от 50 000 до 150 000. Уникальных значений у нас 150 000:

```
SELECT distinct_count(rollup(logs))
FROM (
    (SELECT hyperloglog(4096, v::text)
logs FROM generate_series(1, 100000) v)
    UNION ALL
    (SELECT hyperloglog(4096, v::text)
FROM generate_series(50000, 150000) v)
) hll;
```

Получим вывод с небольшим отклонением в 0,368%:

```
distinct_count
----------------
         150552
```

Функция `stderror()` посчитает сравнительное стандартное отклонение:

```
SELECT       stderror(hyperloglog(8192,
data))
  FROM generate_series(1, 100000) data
```

Результатом будет:

```
stderror
----------------------
0.011490485194281396
```

В рамках данной статьи мы не сможем рассмотреть многообразие функций, доступное для работы с Timescale.

E. Amazon Timestream (Amazon)

Amazon предлагают для работы с временными рядами масштабируемую бессерверную базу данных временных рядов – Amazon Timestream. База Amazon Timestream оптимизирована для работы с

устройствами IoT, для получения, сбора и обобщения больших объемов входящих данных – до триллионов событий в день. В Timestream реализована архитектура автоматического масштабирования, предлагающая практически бесконечные размеры. На платформе реализован механизм магнитного хранилища для исторических данных, иными словами база предоставляет оптимизированный уровень хранения.

На рис.6 представлен вариант схемы, по которой данные могут поступать в Amazon Timestream и обрабатываться далее.

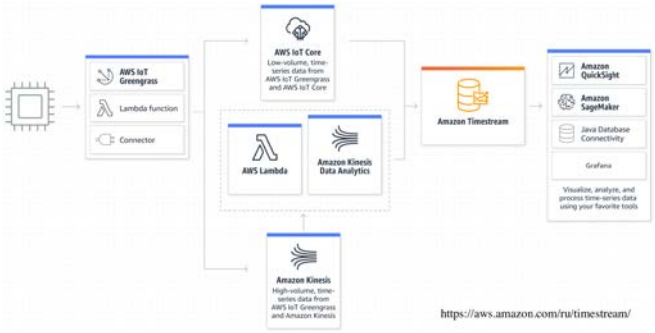

Рис.6 Возможные пути поступления данных в Amazon Timestream и варианты обработки данных внутри сервисов Amazon[12]

Amazon Timestream могут получать данные временных рядов от AWS IoT Core, Amazon Kinesis, Amazon MSK, Telegraph. Визуализировать данные можно с помощью Amazon QuickSight, Grafana и инструментов бизнес-аналитики через JDBC. Также можно интегрировать Timestream с Amazon SageMaker для машинного обучения.

В функционал заложены встроенные функции аналитики для работы с трендами и шаблонами в режиме реального времени. Анализ данных предлагают осуществлять с помощью SQL. Для свежих данных оптимизированы быстрые запросы для определенного времени, в то время как к историческим данным оптимизирован механизм быстрых аналитических запросов. Специально разработанный механизм запросов позволяет получать доступ к последним и историческим данным и анализировать их с помощью одного запроса. С помощью запланированных запросов можно повысить производительность и снизить затраты. Пользователь определяет запросы, которые агрегируют данные, создают сводные таблицы и делают другую аналитику в режиме реального времени. Timeseries автоматически осуществляет эти запросы и записывает результаты в настраиваемую целевую таблицу. Можно настроить информационные панели, отчеты, приложения и системы мониторинга так, чтобы они просто запрашивали таблицы назначения вместо больших исходных таблиц. Это значительно снижает затраты производительной мощности и увеличивает эффективность, скорость работы. Можно хранить данные в таблицах назначения значительно дольше с меньшими затратами, чем при хранении исходной таблицы. Также можно дополнительно оптимизировать затраты на хранение исходных таблиц, сократив срок хранения данных в них.

Также обеспечена безопасность данных: все данные автоматически шифруются, находясь в базе и на пути передачи. Timestream позволяет защищать данные временных рядов благодаря интеграции с AWS Backup. Используя управляемую функциональность этого сервиса, можно создавать неизменяемые резервные копии, автоматизировать управление жизненным циклом копий и копировать такие копии между учетными записями AWS и регионами, настроить периодическое резервное копирование. Первая резервная копия вашей таблицы является полной, а последующие резервные копии той же таблицы копируют только изменения, что делает защиту данных гибкой и экономичной. Для обеспечения безопасности данных магнитного хранилища можно указать клиентский управляемый ключ AWS KMS(CMK, customer managed key). Можно также настроить политики хранения приемлемым для конкретных данных образом, задав когда данные будут автоматически уходить в магнитное хранилище и удаляться оттуда, можно создавать планы резервного копирования, настраивать политики их хранения, задающие по истечении какого срока резервные данные будут переведены в холодное хранилище [12][32].

В процессе получения и обработки временных рядов не последнюю роль может играть Amazon AWS Lambda – сервис для работы с данными, поступающими в реальном времени и, в частности, нестабильно/непостоянно поступающими данными (например, за полдня ничего и вдруг месячный объем за одну минуту), что характеризует работу со многими IoT-устройствами, фиксирующими непредсказуемые изменения внешней среды. В 2014 году, благодаря запуску Lambda, AWS заявляют, что первыми начали работать в сфере бессерверных вычислений, то есть дали возможность клиентам запускать код без приобретения серверов. Поскольку сервис работает без реального «железа», он позволил пользователям сосредоточиться на работе с приложениями, избавляя от необходимости обслуживания серверов. Лямбда запускает экземпляры функции для обработки событий, которую можно вызвать напрямую, используя Lambda–API, или использовать ресурс AWS. Среда выполнения преобразует событие в объект и передает его в функциональный код, также обеспечивая поддержку жизненного цикла для среды выполнения функции и связанных внешних расширений. Каждая фаза начинается с события, которое Lambda отправляет во время выполнения всем зарегистрированным расширениям. Когда AWS вызывает функцию, определяется форма события. Событие представляет из себя документ формата JSON, содержащий данные для обработки лямбда-функций. Пример пользовательских данных, описывающих наш стандартный набор показателей датчиков в помещении:

```
{
 "Temperature": 22.2,
 "Humidity": 45%,
 "Co": 0.2
```

```
}
```

Чтобы указать среде выполнения, что лямбда-выражение должно передавать ответы вашей функции в потоковом режиме, нужно дополнить функцию `streamify Response()`. Функция, переданная в `streamify Response()`, должна принимать параметры: событие (информация о событии вызова URL-адреса функции, такую как HTTP-метод, параметры запросов и текст запроса), ResponseStream (поток, доступный для записи), контекст(предоставляет информацию о вызове, функции, среде выполнения). Объект потока ответов представляет собой доступный для записи поток Node.js.

Можно настроить страницу мониторинга приложения, добавив одну или несколько информационных панелей Amazon CloudWatch в шаблон приложения с типом ресурса AWS::CloudWatch::Dashboard. Пример создания панели мониторинга с одним виджетом, который отображает количество вызовов для функции my-function:

```
Resources:
 MyDashboard:
 Type: AWS::CloudWatch::Dashboard
 Properties:
 DashboardName: my-dashboard
 DashboardBody: |
 {"widgets": [{ "type": "metric",
"width": 12, "height": 6, "properties":
{"metrics": [[
"AWS/Lambda","Invocations","FunctionNam
e", "my-function", {"stat": "Sum",
"label": "MyFunction"}],
 [{ "expression": "SUM(METRICS())",
"label": "Total Invocations"}]],
 "region": "us-east-1", "title":
"Invocations", "view": "timeSeries",
"stacked": false}}]}
```

Когда лямбда-функция вызывается событиями асинхронно, она ставит эти события в очередь, возвращая подтверждение успешного получения события без дополнительной информации. Отдельно процесс считывает события из очереди и отправляет их в функцию. Чтобы вызов функции работал асинхронно, необходимо установить параметр типа вызова Event, как в примере для my-function:

```
aws lambda invoke \
 --function-name my-function \
 --invocation-type Event \
 --cli-binary-format raw-in-base64-out
\
 --payload '{"key": "value"}'
response.json
```

Сервис AWS Lambda и другие инструменты AWS дают широкие возможности для работы с потоками данных временных рядов, которые мы, при всем желании, не сможем уместить в материал данной статьи [25].

Положительным доводом в пользу выбора Amazon Timestream является большой опыт команды Amazon, — 16 лет работы с облачными решениями, — и статус Amazon, как одного из лидеров сферы облачных технологий. Уровень безопасности AWS, как заявлено у поставщика услуг, позволяет удовлетворить требования даже международных банков и учреждений в сфере обороны, а, как мы знаем, важность безопасности данных для IoT сложно переоценить. У AWS заявлено более 300 сервисов по обеспечению безопасности, 98 стандартов безопасности и сертификаций на соответствие требованиям и 117 сервисов для хранения клиентских данных с услугами по их шифрованию. Клиентами AWS Timestream являются такие компании, как Trimble, поставщик технологических решений для строительной, транспортной отраслей, коммуникационная платформа PunNub и Autodesk, мировой поставщик программного обеспечения для архитекторов, инженеров и медиа [12].

Ж.OpenTSDB

OpenTSDB включает в свою структуру TSD(демон временных рядов) и набор утилит командной строки. Каждый TSD является независимым, отсутствует иерархия. Еще одна достойна по уровню и масштабам для работы с BigData база. Взаимодействие с OpenTSDB преимущественно осуществляется за счет запуском одного или нескольких TSDs, каждый из которых использует HBase[33] или Google Bigtable (Google Cloud) сервис для хранения и извлечения временных рядов (рис.7).

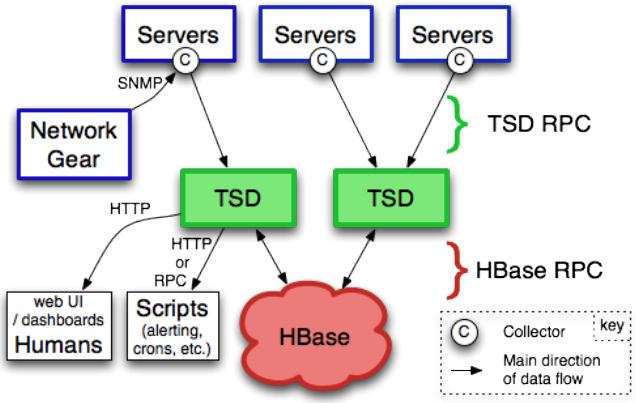

Рис.7 Структура сервиса OpenTSDB[9]

Алгоритм оптимизирован для быстрой агрегации схожих временных рядов с целью минимизации размера хранилища. Метки позволяют отделять похожие точки данных из разных источников или связанных объектов. В сравнении с Gorilla, например, где идентификация временных рядов осуществлялась с помощью единственного строкового ключа и полагалась на инструменты более высокого уровня для извлечения и идентификации метаданных временных рядов, идентификация в OpenTSDB имеет более богатую модель данных [16]. Приведем пример нескольких единиц данных в OpenTSDB:

```
mysql.flat50 1287333217 327810227706
schema=foo host=db1
mysql.flat49 1287333217 6604859181710
```

```
schema=foo host=db1
mysql.flat50  1287333232  327812421706
schema=foo host=db1
mysql.flat49  1287333232  6604901075387
schema=foo host=db1
mysql.flat50  1287333321  340899533915
schema=foo host=db2
mysql.flat49  1287333321  5506469130707
schema=foo host=db2
```

В примере выше представлено 6 различных единиц данных, которые относятся к 4 разным временным рядам. Каждое сочетание метрики и метки относится к другому временному ряду. Ряды относятся к одной из двух меток – кухня или гостиная. Каждый временной ряд для метрики должен иметь такое же количество тэгов, как и другие ряды данной метрики. Не рекомендуется более 6-7 тегов у одной единицы данных, так как затраты, связанные с хранением новых единиц данных зависимы от количества тегов, превышающих эту цифру. С метками, как в примере, будет легко создавать диаграммы и панели, демонстрирующие сетевую активность MySQL для каждого хоста и/или для каждой схемы.

Пользователи TSD никогда не нуждаются в доступе к базовым хранилищам напрямую. Сервис предоставляет услугу вечного хранения необработанных данных, что может быть весьма ценно для будущих планов по детальной их аналитике и рассмотрении «под новым углом». Платформа дает возможности масштабирования до нескольких миллионов операций в секунду.

Единица данных в OpenTSDB характеризуется набором параметров: название метрики, UNIX временная метка (секунды или миллисекунды), значение (целочисленное значение, число с плавающей точкой, событие формата JSON или гистограмма/дайджест) и метка/сэт меток (пары ключ+значение, обозначающие временной ряд, которому принадлежит единица). Комбинация метрики и метки характеризует каждый временной ряд. Распространенный вариант использования тэгов – аннотирование единиц данных названием машины, которая их сгенерировала, а также названием кластера/пула, к которому принадлежит машина. Это позволяет легко создавать информационные панели, отображающие состояние сервиса для каждого сервера, а также панели, отображающие агрегированное состояние по логическим пулам серверов.

Также OpenTSDB предлагают встроенный инструмент построения линейных графов на основе нескольких метрик и меток. В доступе существует HTTP API для привязки к внешним системам мониторинга, панелям, статистическим пакетам и средствам автоматизации.

С OpenTSDB 2.0 добавлена опция хранения нечисловых аннотаций вместе с единицами данных, позволяющая отслеживать метаданные, качественные метрики и другие типы информации. На момент написания данного материала последняя действующая версия OpenTSDB 2.4. На сайте сообщества написано, что к выходу запланирована 3-я версия платформы OpenTSDB, в которой реализуют функцию распределения запросов между несколькими TSD для повышения пропускной способности, разделенное по времени кэширование запросов с целью их улучшения, возможность использования выражений `GROUP BY`, понижающей дискретизации, арифметических модификаций в любом порядке и возможная поддержка UDF, интеграция с библиотеками моделирования (например, EGADs) для более глубокого анализа временных рядов. Также разработчики пишут о своей готовности добавить фичи, которые нужны пользователям, примут запросы и предложения [33][9].

3. CLICK HOUSE (YANDEX)

Колоночная база данных с открытым кодом для сбора данных в реальном времени и аналитики от Яндекса. Аналитический инструмент Яндекса, который заслужил популярность не только в России, – аналитическая колоночная СУБД ClickHouse. В 2016 г. Яндекс открыли исходный код для ClickHouse, расширив ее возможности и популярность. Платформа позволяет выполнять аналитические запросы в реальном времени на структурированных больших данных. Как отмечают на сайте платформы, колоночные СУБД лучше всего работает со сценариями OLAP и обрабатывает большинство запросов «как минимум в 100 раз быстрее». Так как значения колонки/столбца хранятся физически рядом, соответственно более доступны для прохода и отсутствует необходимость в чтении всей таблицы для извлечения необходимых данных по большинству типовых запросов [26]. ClickHouse используют такие компании как Bloomberg, Cloudflare, Тинькофф банк, Авито, Mail.ru, национальный домен верхнего уровня Чили – NIC Labs Chile и не только [20][21].

Поддерживает множество источников данных, инструментов визуализации и ряд языков программирования. Основная задача, на основе решения которой изначально создавалась база, – максимально быстрая фильтрация и агрегирование данных. Быстрота чтения достигается за счет проиндексированных данных и колоночной структуры организации – ресурсы не расходуются на чтение ненужных данных. Также сжатое хранение данных достигается за счет колоночной организации – как правило, данные соседних строк столбцов содержат не сильно отличные друг от друга значения. Также для сжатия данных у сервиса предусмотрены специализированные кодеки. Кодеки используют специфику данных для их сжатия. Некоторые кодеки не сжимают данные как таковые, вместо этого они предварительно обрабатывают данные таким образом, чтобы на втором этапе сжатия с помощью кодека общего назначения можно было достичь более эффективного сжатия. Например, `Delta(delta_bytes)` производит сжатие данных за счет замены значений на разницу соседних значений, за исключением самого первого значения, которое остается неизменным. `DoubleDelta(bytes_size)` делает то же самое, но записывает в компактной двоичной форме. Кодек можно использовать для любых типов значений фиксированной длины. `DoubleDelta(bytes_size)` реализует алгоритм,

используемый в Gorilla TSDB [32], расширяя его для поддержки 64-разрядных типов. Тоже взят у Gorilla алгоритм одноименного кодека Gorilla (bytes_size), который вычисляет XOR между текущим и предыдущим значениями с плавающей запятой и записывает данные в двоичном формате. Чем меньше разница между последовательными значениями, т.е. чем медленнее изменяются значения ряда, тем выше степень сжатия. Еще один кодек для подготовки данных – GDC вычисляет наибольший общий знаменатель для значений столбца и отлично подходит для оптимизации хранения кратных целочисленных значений, а также форматов decimal и date/time. К примеру, когда база получает данные от датчика по графику в заданный промежуток времени. Кодек FPC (level, float_size) предсказывает следующее значение с плавающей запятой, используя лучший из двух предикторов, затем применяет XOR для предсказанного значения и фактического и ведущий ноль сжимает результат. Кодек T64 обрезает неиспользуемые старшие разряды в целочисленных типах данных. На каждом шаге алгоритма кодек берет блок из 64 значений, помещает их в битовую матрицу размером 64x64, транспонирует ее, обрезает неиспользуемые биты и возвращает остальные в виде последовательности. Помимо сжатия данных есть кодеки шифрования – AES_128_GCM_SIV и AES-256-GCM-SIV.

ClickHouse не только хранит данные в столбцах, но и обрабатывает данные по столбцам. Это приводит к лучшему использованию кэш процессора и позволяет использовать инструкции SIMD CPU, – когда процессор распараллеливает входящие потоки инструкций по разным вычислительным блокам. Даже для выполнения одного запроса ClickHouse в случае необходимости может использовать все доступные процессорные ядра и диски, не только на одном сервере, но и на всех процессорных ядрах и дисках кластера.

ClickHouse поддерживает декларативный язык запросов, основанный на SQL и во многих случаях идентичный стандарту ANSI SQL. В запросах используют `GROUP BY, ORDER BY,` подзапросы в `FROM`, операторов `JOIN, IN`, функции окон и скалярные подзапросы. Запрос по созданию таблицы flat50 в БД house_control может выглядеть так:

```
 CREATE TABLE [IF NOT EXISTS]
[house_control.]flat50 [ON CLUSTER
cluster]
(temperature [float] [NULL|NOT NULL]
[DEFAULT|MATERIALIZED|EPHEMERAL|ALIAS
expr1] [compression_codec] [TTL expr1]
[COMMENT 'comment for column'],
humidity [float] [NULL|NOT NULL]
[DEFAULT|MATERIALIZED|EPHEMERAL|ALIAS
expr2] [compression_codec] [TTL expr2]
[COMMENT 'comment for column'],
...
) ENGINE = engine
COMMENT 'comment for table'
```

В качестве `ENGINE` задается движок таблицы, например, семейство движков Merge Tree (деревья слияния), семейство Log, интегрированные и специальные. Движки также задаются для базы данных: Atomic, Lazy, Replicated, PostgreSQL, MySQL, SQLite и экспериментальные MaterializedPostgreSQL и MaterializedMySQL. Для разных движков могут работать и не работать конкретные выражения запросов, что нужно учитывать при создании таблицы. Движки семейства Merge Tree – наиболее надежный механизм Click House и предназначены для вставки очень большого объема данных в таблицу. Данные быстро записываются в таблицу по частям, затем применяются правила для объединения частей в фоновом режиме. Опционально в описании столбца можно указать значение по умолчанию: `DEFAULT` (значение по умолчанию), `MATERIALIZED` (рассчитываемое значение по умолчанию), `EPHEMERAL` (не хранятся в таблице, задается для возможности создания из них выражений значений по умолчанию для других столбцов) или `ALIAS`(не хранятся в таблице, когда запрос `SELECT` обращается к значению такого столбца, значение вычисляется во время запроса). Также для столбцов таблиц можно установить `PRIMARY KEY` и `CONSTRAINTS`. В таблицах семейства Merge Tree можно устанавливать время хранения значений с помощью выражения TTL.
Выражение `CHECK TABLE` дает возможность проверить, не повреждены ли данные в таблице, путем сравнения фактических размеров файлов с ожидаемыми значениями, которые хранятся на сервере. Например:

```
CHECK TABLE [house_control.]flat50
[PARTITION partition_expr]
```

Если размеры файлов не соответствуют сохраненным значениям, данные повреждены. Такое может случиться, например, при сбое системы во время выполнения запроса. Результатом запроса `CHECK TABLE` будет результирующий столбец с одной строкой, содержащей значения логического типа:
0 – Данные повреждены
1 – Данные сохраняют целостность

Задача выражения `OPTIMIZE` – попытка инициализировать незапланированное слияние частей данных в таблицах. Запрос будет выглядеть так:

```
OPTIMIZE TABLE [house_control.]flat50
[ON CLUSTER cluster] [PARTITION
partition | PARTITION ID
'partition_id'] [FINAL] [DEDUPLICATE
[BY expression]]
```

В `PARTITION` опционально указывают конкретный раздел, который должен быть оптимизирован. Если указать `FINAL`, оптимизация выполняется даже тогда, когда все данные уже находятся в одной части. Можно управлять этим поведением с помощью `optimize_skip_merged_partitions`. Кроме того, слияние будет выполнено принудительно, даже если выполняются параллельные слияния. Если указать `DEDUPLICATE`, будут дедуплицированы идентичные строки.
На основе только части инструментов, которые мы рассмотрели, для работы с данными в ClickHouse,

можно сделать вывод, что широкий спектр инструментов и механизмов может оказаться весьма полезным для работы с гетерогенными данными временных рядов [44].

И. MongoDB

MongoDB и MongoDB Atlas, мульти облачный сервис для развертывания и управления БД, называют себя «идеальными партнерами для любого развертывания IoT» (рис.8). Стоит в самом начале отметить, что в 2022 году MongoDB заявили об удалении всех данных пользователей из России и Беларуси и использовать сервис сегодня с помощью «обходных» схем в этих государствах небезопасно для ваших данных.
Пользователям предлагают сервис развертывания с использованием ПО на устройстве клиента, «в полях» и облачный сервис. Доступна работа в мульти-облачном формате с подключением AWS, Azure, Google Cloud.

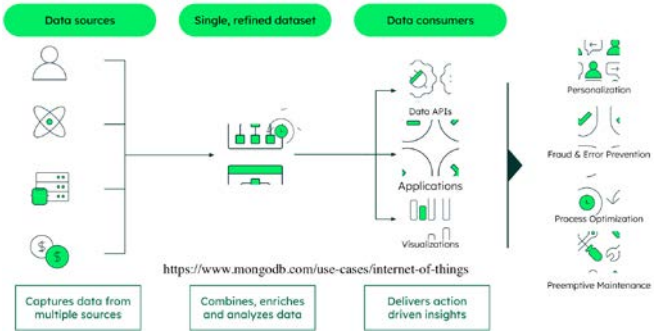

Рис.8 Схема получения данных временных рядов интернета вещей MongoDB и варианты дальнейшего применения[28]

Сервис имеет гибкость к частым изменениям, дополнениям, а также возможность смешивать разные данные — операционные и временные ряды. База оптимизирована для предоставления аналитики в режиме реального времени, автоматически распределяет входящие данные по уровням. Что является несомненным плюсом, платформа разработчика MongoDB поддерживает полный цикл работы с IoT, от приема данных, распределения, работы с запросами, аналитикой и визуализацией данных, до онлайн архивации, исключая на стороне пользователя сложности по синхронизации разных сервисов и снижает риск утечки данных на одном из этапов.
Коллекции временных рядов (buckets), или группы документов, имеют индексы. Благодаря тому, что индексируются группы, а не отдельные документы, количество индексов сильно сокращается. Кластеризованные индексы упорядочивают данные на диске по времени. Все это позволяет ускорить доступ к данным. Таким образом, единственный индекс "meta" характеризует группу документов.
Чтобы создать коллекцию данных временных рядов используют опцию timeseries:

```
db.createCollection("weather", {
timeseries: {
timeField: "timestamp",
metaField: "sensorID",
granularity: "minutes"
},
expireAfterSeconds: 9000
})
```

Опция metaField представляет из себя ярлык или тэг, который является уникальным для каждого временного ряда и никогда не изменяется, если его не изменить своими руками. Для добавления единичного события, т.е. одного документа, используют `db.collection.insertOne()`. Пример добавления нескольких событий, где *sensorId* представляет из себя *metaField*, как мы и указали при создании коллекции в коде выше, выглядит так:

```
db.weather.insertMany( [
{
"sensorId": 734,
"timestamp":          ISODate("2023-07-
03T21:10:00.000Z"),
"temperature": 22.4
},
{
"sensorId": 562,
"timestamp":          ISODate("2023-07-
03T21:10:00.000Z"),
"temperature": 22.1
},
{
"sensorId": 734,
"timestamp":          ISODate("2023-07-
03T21:40:00.000Z"),
"temperature": 22.4
},
{
"sensorId": 562,
"timestamp":          ISODate("2023-07-
03T21:40:00.000Z"),
"temperature": 22.1
},
```

Опция `expireAfterSeconds` дает возможность заменять TTL индексы – специальные индексы, которые база данных использует для автоматического удаления документов из коллекции по истечении определенного промежутка времени или в определенное тактовое время.
Степень детализации (granuality) задает период времени в который измерения с одинаковым metaField хранятся и группируются вместе. Это могут быть секунды (высокая детализация), минуты или часы (низкая детализация) и т.д. В примере выше в базу поступили четыре измерения температуры с двух сенсоров с промежутком в 30 минут. Данные в MongoDB добавляются по принципу одно события = один документ (как добавить несколько документов мы описали выше), следовательно, мы получили 4 документа для двух временных рядов с уникальным *metaField*, которые группируются в одно хранилище по этому уникальному определителю временного ряда и хранятся в базе уже как один документ (рис.9).

Рис.9 Группировка данных для хранения в MongoDB[27]

Запрос к временным рядам по поиску данных сенсора с ID 734 и времени замера, указанном в скобках, может выглядеть так:

```
db.weather.find(
"sensorId": 734,
"timestamp": {$gte:
ISODate("2023-07-03T21:40:000Z")}
}
})
```

Чтобы найти один документ в коллекции, используйте:

```
db.weather.findOne({
   "timestamp":       ISODate("2021-05-
18T00:00:00.000Z")
})
```

При запросе происходят два ключевых процесса: распаковка сегмента(bucket) и поиск данных, удовлетворяющих всем критериям запроса. Запрос на агрегирование может выглядеть так:

```
db.weather.aggregate( [
   {$project: {
         date: {$dateToParts: { date:
"$timestamp" }},
         temp: 1}},
   {$group:    {_id:    {date:    {year:
"$date.year",
               month: "$date.month",
               day: "$date.day"}},
         avgTmp: { $avg: "$temp" }}}] )
```

Пользуются сервисом MongoDB для работы с IoT такие мировые бренды, как Bosch, Vaillant, SoftwareAG, Toyota, Mercedes-Benz, Vodafone. Комментарии клиентов по использованию платформы можно прочесть в блоге на официальном сайте MongoDB [27][28].

V. Заключение

Строго говоря, ключ к созданию универсального сервиса для работы с базами данных кроется в оптимизации работы сервиса с разнородными временными рядами. Как в своей работе, опубликованной издательством МГУ еще в 1991 году, писал кандидат экономических наук, экономист-математик Сергей Владиславович Смирнов: «Можно ли, отказавшись от средств определения структуры данных, претендовать на то, чтобы некая программа могла обслуживать доступ к разным БД? Ответ простой: можно, если удастся предложить универсальную схему описания экономических временных рядов. Ясно, что единственного правильного решения здесь не существует. Ясно также, что самый простой вариант – ограничиться одним полем, в котором можно записать произвольную информацию, – заведомо не подходит: в этом случае нельзя предложить удобной процедуры автоматизированного запроса» [1].

Работа по оптимизации баз данных временных рядов критически важна для набирающих обороты процессов IoT и неизбежно такая работа будет продолжаться. На сегодняшний момент рынок предлагает далеко не одно решение с гибким масштабированием, но взрывающих рынок инноваций в работе с классическими колоночными БД, хранящими временные ряды, не было со времен появления такой структуры. Разница в базах данных часто ограничивается синтаксисом запросов и максимальным его упрощением для пользователя, что, несомненно, вносит вклад в популяризацию конкретного продукта, но не решает на инновационном уровне главной задачи хранения все больших и больших объемов данных в условиях ограниченной физической памяти. А также производительность самых мощных серверов растет медленнее, чем объемы данных. И эта задача должна решаться в настоящий момент, иначе процессы развития технологий остановятся, и возникнет коллапс данных.

Все базы, ориентированные на хранение временных рядов и рассмотренные в данной работе, так или иначе, учитывают необходимость масштабирования. Такие крупные и популярные платформы как Click House, Amazon, Influxdata, KDB+ и Timescale предлагают гибкое масштабирование. Важным нюансом при выборе базы данных является развитость ее сервисов, экосистемы и интегрируемость с другими платформами. В данном вопросе, на основе проанализированных данных, выигрывают TimeScale, OpenTSDB, Prometheus и ClickHouse. Сравнительно большее богатство данных базы временных рядов предлагают Influxdata, OpenTSDB и Click House. Высочайшую скорость работы анонсируют Click House, KDB+, OpenTSDB и Timescale. Удобство работы с запросами для аналитиков, не погружаясь в программирование, предлагают OpenTSDB, Influxdata, Timescale и собственно сервис аналитики Graphite. В целом, встроенные сервисы аналитики есть в том или ином виде у всех поставщиков услуг. Максимальное сжатие данных предлагают Timescale, KX Systems, Graphite с The Whisper, OpenTSDB. О безопасности и сохранности данных «громче» всех заявляют Amazon, Timescale и Click House.

В заключении хочется отметить, что выбор базы данных и СУБД для работы с временными рядами строго индивидуален и зависит от множества факторов, поэтому дать универсальные рекомендации в пользу конкретного сервиса на практике

невозможно.

Благодарности

Хочу выразить благодарность руководству факультета ВМК МГУ им. Ломоносова за предоставленную возможность профессиональной переподготовки и, в отдельности, старшему научному сотруднику лаборатории математической физики Барашкову Игорю Сергеевичу и ведущему научному сотруднику лаборатории открытых информационных технологий Намиот Дмитрию Евгеньевичу за интересный и обширный курс по базам данных и работе с ними.

Библиография

[1] Смирнов, Сергей Владиславович “Базы данных временных рядов.Пакет Data Guide”:Учеб.-метод.пособие / Отв.ред.М.И.Лугачев; "Всесоюз.экон.об-во. Центр “Экономическая кибернетика”,Экон.фак.и др. М. : Изд-во Моск.ун-та, 1991. с.7

[2] Бриллинджер, Давид Р. “Временные ряды : Обработка данных и теория” / Д.Р. Бриллинджер ; Пер. с англ. А.В. Булинского, И.Г. Журбенко ; Под ред. А.Н. Колмогорова М. : Мир, 1980. - С.7, 13, 19, 49.

[3] Григорьев, Юрий Александрович “Реляционные базы данных и системы NoSQL” : учеб. пособие. / Григорьев Ю. А., Плутенко А. Д., Плужникова О. Ю.. - Благовещенск : Амур. гос. ун-т, 2018. -С. 19, 304-307, 323, 331-337.

[4] Архипенков, Сергей Яковлевич. “Хранилища данных : От концепции до внедрения” / С.Архипенков,Д.Голубев,О.Максименко;Под.ред.С.Архипенкова М. : Диалог-МИФИ, 2002. 1: с.23-25

[5] Намиот Д. Е. Базы данных временных рядов и средства обработки //Актуальные проблемы системной и программной инженерии: Материалы. – 2015. – С. 151-158.

[6] Официальный сайт: https://www.influxdata.com/ Дата доступа: 18.07.23

[7] DB Engines Официальный сайт: https://db-engines.com/en/ranking/time+series+dbms Дата доступа: 2.10.23

[8] Kdb Официальный сайт: https://code.kx.com/q4m3/14_Introduction_to_Kdb%2B/ Дата доступа: 18.08.23

[9] OpenTSDB Официальный сайт: http://opentsdb.net/overview.html Дата доступа: 29.07.23

[10] Prometeus Официальный сайт: https://prometheus.io/docs/prometheus/latest/storage/ Дата доступа: 2.08.23

[11] TimeScale Официальный сайт: https://www.timescale.com/products Дата доступа: 12.08.23

[12] Amazon S3 Официальный сайт: https://aws.amazon.com/ru/s3/ ,Дата доступа: 16.08.23

[13] Daily U.S Equity Matched Volumes (millions, includes CS2), NYSE Доступно: https://www.nyse.com/markets/us-equity-volumes Дата доступа: 16.03.23

[14] PostgreSQL, The Time-Series Database You Want by Chris Engelbert. Доступно: https://www.youtube.com/watch?v=k1xVlS6hdxU Дата доступа: 20.08.23

[15] PostgreSQL Официальный сайт: https://www.postgresql.org/ Дата доступа: 18.08.23

[16] T. W. Wlodarczyk. Overview of Time Series Storage and Processing in a Cloud Environment. In Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), pages 625–628. IEEE Computer Society, 2012

[17] Официальный сайт: https://graphiteapp.org/ Дата доступа: 24.08.23

[18] Alternatives to Time Series Databases, HAZELCAST. Доступно: https://hazelcast.com/glossary/time-series-database/#:~:text=Relational%20database%20management%20systems%20 Дата доступа: 02.09.23

[19] Презентация “Prometheus как time series database”, Тимур Нурутдинов (Lamoda) Доступно: https://youtu.be/5g6NAKzTKTs Дата доступа: 15.05.23

[20] Yandex Cloud Официальный сайт: https://cloud.yandex.com/en/services/ydb Дата доступа: 15.09.23

[21] Yandex Clickhouse Официальный сайт: https://yandex.ru/dev/clickhouse/ Дата доступа: 15.09.23

[22] RAID-массивы, studme.org Доступно: https://studme.org/97217/informatika/raid-massivy Дата доступа: 23.09.23

[23] Официальная документация: https://graphite.readthedocs.io/en/latest/whisper.html , https://graphite.readthedocs.io/en/latest/functions.htm Дата доступа: 23.08.23

[24] “Модель Хольта-Уинтерса: Математические аспекты и компьютерная реализация” © 2016 М.Г. Семененко, кандидат физико-математических наук, доцент кафедры «Высшая математика» Л.А. Унтилова, старший преподаватель кафедры «Экономика и организация производства» Калужский филиал Московского государственного технического университета имени Н.Э. Баумана, Калуга (Россия) Доступно: https://www.vektornaukieconomika.ru/jour/article/view/208/200 Дата доступа: 8.10.23

[25] AWS Официальная документация: https://docs.aws.amazon.com/pdfs/lambda/latest/dg/lambda-dg.pdf Дата доступа: 7.09.23

[26] Clickhouse SQL Официальный сайт: https://clickhouse.com/docs/en/sql-reference Дата доступа: 7.09.23

[27] MongoDB Официальная документация: https://www.mongodb.com/docs/v7.0/core/timeseries/timeseries-procedures/#std-label-manual-timeseries-collection-create Дата доступа: 10.09.23

[28] Официальный сайт: https://www.mongodb.com/use-cases/internet-of-things Дата доступа: 2.09.23

[29] Harizopoulos S., Abadi D.J., Madden S., Stonebraker M. OLTP through the looking glass, and what we found there // Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker / Ed. by Brodie M.L. ACM / Morgan & Claypool, 2019. P. 409–439. DOI: 10.1145/3226595.3226635.

[30] Queiroz-Sousa P.O., Salgado A.C. A review on OLAP technologies applied to information networks // ACM Trans. Knowl. Discov. Data. 2020. Vol. 14, no. 1. P. 8:1–8:25. DOI: 10.1145/3370912.

[31] Иванова Е.В., Цымблер М.Л. “Обзор современных систем обработки временных рядов” // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2020. Т. 9, No 4. С. 79–97. DOI: 10.14529/cmse200406.

[32] «Gorilla: A Fast, Scalable, In-Memory Time Series Database» Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, Kaushik Veeraraghavan Доступно: http://www.vldb.org/pvldb/vol8/p1816-teller.pdf Дата доступа: 8.10.23

[33] HBASE Официальный сайт: http://hbase.org/ Дата доступа: 2.10.23

[34] HighCharts https://www.highcharts.com/ Дата доступа: 2.10.23

Статья получена 13.10.23

С.В.Гаврикова окончила Московский Государственный Университет Печати им.И.Федорова, слушатель курса “Разработчик профессионально ориентированных компьютерных технологий” ВМК МГУ им. Ломоносова, индивидуальный предприниматель в сфере диджитал маркетинга (e-mail: svetagavrikova@gmail.com)

Time series databases overview

S.V. Gavrikova

***Abstract* — This article examines time series as a data type, its use in the Internet of Things (IoT), and the most popular time series storage platforms (time series databases). Currently, the Internet of Things is in dire need of efficient and optimized time series databases and management systems to deal with heterogeneous data. Time series database management systems (DBMS) are the second most popular category on db-engines.com. The basis for the selection of databases for analysis was mainly the top part of the db-engines.com DBMS rating. The article discusses database structures, data aggregation capabilities, scaling flexibility, storage optimization, security, the ability to use auxiliary tools, query syntax, including examples with time series obtained from sensors. The main goal of this work is to study real-life platforms with their pros and cons, highlight the need for an informed approach to choosing a tool, as well as the critical importance of finding innovative technologies for storing time series for the future of the entire digital society.**

Keywords — time series, time series database, time series storage, influxdata, clickhouse, mongodb, opentsdb, timescale, kdb, prometheus, graphite, amazon timestream

[1] Smirnov, Sergej Vladislavovich “Bazy dannyh vremennyh rjadov.Paket Data Guide”:Ucheb.-metod.posobie / Otv.red.M.I.Lugachev; "Vsesojuz.jekon.ob-vo. Centr “Jekonomicheskaja kibernetika”,Jekon.fak.i dr. M. : Izd-vo Mosk.un-ta, 1991. c.7

[2] Brillindzher, David R. “Vremennye rjady : Obrabotka dannyh i teorija” / D.R. Brillindzher ; Per. s angl. A.V. Bulinskogo, I.G. Zhurbenko ; Pod red. A.N. Kolmogorova M. : Mir, 1980. -S.7, 13, 19, 49.

[3] Grigor'ev, Jurij Aleksandrovich “Reljacionnye bazy dannyh i sistemy NoSQL” : ucheb. posobie. / Grigor'ev Ju. A., Plutenko A. D., Pluzhnikova O. Ju.. - Blagoveshhensk : Amur. gos. un-t, 2018. -S. 19, 304-307, 323, 331-337.

[4] Arhipenkov, Sergej Jakovlevich. “Hranilishha dannyh : Ot koncepcii do vnedrenija” / S.Arhipenkov,D.Golubev,O.Maksimenko;Pod.red.S.Arhipenkova M. : Dialog-MIFI, 2002. 1: c.23-25

[5] Namiot D. E. Bazy dannyh vremennyh rjadov i sredstva obrabotki //Aktual'nye problemy sistemnoj i programmnoj inzhenerii: Materialy. – 2015. – S. 151-158.

[6] Oficial'nyj sajt: https://www.influxdata.com/ Data dostupa: 18.07.23

[7] DB Engines Oficial'nyj sajt: https://db-engines.com/en/ranking/time+series+dbms Data dostupa: 2.10.23

[8] Kdb Oficial'nyj sajt: https://code.kx.com/q4m3/14_Introduction_to_Kdb%2B/ Data dostupa: 18.08.23

[9] OpenTSDB Oficial'nyj sajt: http://opentsdb.net/overview.html Data dostupa: 29.07.23

[10] Prometeus Oficial'nyj sajt: https://prometheus.io/docs/prometheus/latest/storage/ Data dostupa: 2.08.23

[11] TimeScale Oficial'nyj sajt: https://www.timescale.com/products Data dostupa: 12.08.23

[12] Amazon S3 Oficial'nyj sajt: https://aws.amazon.com/ru/s3/ ,Data dostupa: 16.08.23

[13] Daily U.S Equity Matched Volumes (millions, includes CS2), NYSE Dostupno: https://www.nyse.com/markets/us-equity-volumes Data dostupa: 16.03.23

[14] PostgreSQL, The Time-Series Database You Want by Chris Engelbert. Dostupno: https://www.youtube.com/watch?v=k1xVlS6hdxU Data dostupa: 20.08.23

[15] PostgreSQL Oficial'nyj sajt: https://www.postgresql.org/ Data dostupa: 18.08.23

[16] T. W. Wlodarczyk. Overview of Time Series Storage and Processing in a Cloud Environment. In Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), pages 625–628. IEEE Computer Society, 2012

[17] Oficial'nyj sajt: https://graphiteapp.org/ Data dostupa: 24.08.23

[18] Alternatives to Time Series Databases, HAZELCAST. Dostupno: https://hazelcast.com/glossary/time-series-database/#:~:text=Relational%20database%20management%20systems%20 Data dostupa: 02.09.23

[19] Prezentacija “Prometheus kak time series database”, Timur Nurutdinov (Lamoda) Dostupno: https://youtu.be/5g6NAKzTKTs Data dostupa: 15.05.23

[20] Yandex Cloud Oficial'nyj sajt: https://cloud.yandex.com/en/services/ydb Data dostupa: 15.09.23

[21] Yandex Clickhouse Oficial'nyj sajt: https://yandex.ru/dev/clickhouse/ Data dostupa: 15.09.23

[22] RAID-massivy, studme.org Dostupno: https://studme.org/97217/informatika/raid-massivy Data dostupa: 23.09.23

[23] Oficial'naja dokumentacija: https://graphite.readthedocs.io/en/latest/whisper.html , https://graphite.readthedocs.io/en/latest/functions.htm Data dostupa: 23.08.23

[24] “Model' Hol'ta-Uintersa: Matematicheskie aspekty i komp'juternaja realizacija” © 2016 M.G. Semenenko, kandidat fiziko-matematicheskih nauk, docent kafedry «Vysshaja matematika» L.A. Untilova, starshij prepodavatel' kafedry «Jekonomika i organizacija proizvodstva» Kaluzhskij filial Moskovskogo gosudarstvennogo tehnicheskogo universiteta imeni N.Je. Baumana, Kaluga (Rossija) Dostupno: https://www.vektornaukieconomika.ru/jour/article/view/208/200 Data dostupa: 8.10.23

[25] AWS Oficial'naja dokumentacija: https://docs.aws.amazon.com/pdfs/lambda/latest/dg/lambda-dg.pdf Data dostupa: 7.09.23

[26] Clickhouse SQL Oficial'nyj sajt: https://clickhouse.com/docs/en/sql-reference Data dostupa: 7.09.23

[27] MongoDB Oficial'naja dokumentacija: https://www.mongodb.com/docs/v7.0/core/timeseries/timeseries-procedures/#std-label-manual-timeseries-collection-create Data dostupa: 10.09.23

[28] Oficial'nyj sajt: https://www.mongodb.com/use-cases/internet-of-things Data dostupa: 2.09.23

[29] Harizopoulos S., Abadi D.J., Madden S., Stonebraker M. OLTP through the looking glass, and what we found there // Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker / Ed. by Brodie M.L. ACM / Morgan & Claypool, 2019. P. 409–439. DOI: 10.1145/3226595.3226635.

[30] Queiroz-Sousa P.O., Salgado A.C. A review on OLAP technologies applied to information networks // ACM Trans. Knowl. Discov. Data. 2020. Vol. 14, no. 1. P. 8:1–8:25. DOI: 10.1145/3370912.

[31] Ivanova E.V., Cymbler M.L. “Obzor sovremennyh sistem obrabotki vremennyh rjadov” // Vestnik JuUrGU. Serija: Vychislitel'naja matematika i informatika. 2020. T. 9, No 4. S. 79–97. DOI: 10.14529/cmse200406.

[32] «Gorilla: A Fast, Scalable, In-Memory Time Series Database» Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, Kaushik Veeraraghavan Dostupno: http://www.vldb.org/pvldb/vol8/p1816-teller.pdf Data dostupa: 8.10.23

[33] HBASE Oficial'nyj sajt: http://hbase.org/ Data dostupa: 2.10.23
[34] HighCharts https://www.highcharts.com/ Data dostupa: 2.10.23