Системы хранения и обработки
потоков больших данных

Хранилища потоковых данных.

HDFS, Kafka, Pravega, Ignite Persistence, ...

Самарев Роман Станиславович, 2023

МГТУ им. Н.Э. Баумана
Кафедра Компьютерные системы и сети

План презентации

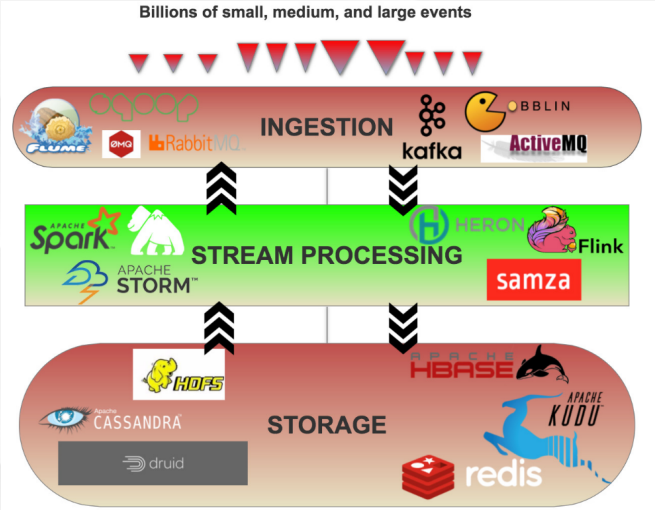
Трёхуровневая модель

O.-C. Marcu, A. Costan, G. Antoniu, M. Pérez-Hernández, R. Tudoran,

S. Bortoli, and B. Nicolae. *Storage and Ingestion Systems in Support of Stream Processing: Survey.*
PhD thesis, INRIA Rennes-Bretagne Atlantique and University of Rennes 1,

France, 2018 O. Marcu. *KerA: A Unified*

Ingestion and Storage System for Scalable Big Data Processing. (KerA : un système unifié d'ingestion et de stockage pour le traitement efficace du Big Data).
PhD thesis, INSA Rennes, France, 2018



Слой приёма ("заглатывания") сообщений (Ingestion)

Обеспечивает поступление, буферизацию, предобработку и фильтрацию сообщений. Обычно, имеет ограничения в части длительности хранения (часы, дни) и ограниченный набор функций доступа к данным без возможности произвольного доступа.

Kafka, Flume, Rabbit MQ, Active MQ, Pravega, Ignite,....

Слой долговременного хранения (Storage)

Предназначен для хранения данных после окончания обработки. Может быть использован для аналитики, пакетно-ориентированной обработки данных. Обеспечивает гарантированное хранение. Принципиально не предназначен для обслуживания запросов длительностью порядка секунды.

HDFS, Cassandra, HBASE, KUDU, Druid...

Слой потоковой обработки (Processing)

Использует сообщения, накопленных Ingestion-слоем. Обеспечивает обработку в режиме реального времени с гарантией сохранения данных.

Области применения. Монетизация видео

Пример - архитектура Dataflow компании Google.

Во время показа видео добавляется контекстная реклама на основе предпочтений. Необходимо собирать как предпочтения для показа видео, так и для показа рекламы.

Особенности:

- События должны быть поглощены и запущены на обработку как можно быстрее
- События и агрегированные результаты сохраняются последующего анализа и обработки
- Пользователи видео- и рекламных платформ требуют статистику и мониторинг в реальном времени.

Сценарий контроля ПО и оборудования:

- Мониторинг процессоров, памяти, дисков, внешний обмен данными с целью контроля работоспособности и уровня загрузки
- Мониторинг приложений и формирование предупреждений и оповещений о необходимых действиях согласно регламенту обслуживания.

Обычно, после выдачи оповещений, собранные данные не хранятся. В отдельных случаях могут храниться агрегаты.

Сценарий мониторинга и предотвращения атак:

- Заголовки сетевых пакетов в реальном времени собираются, индексируются и сохраняются для долговременного хранения
- Системы мониторинга непрерывно выполняют запросы над подвижными и сохранёнными данными, формируя предупреждения об идентифицированных пробелемах
- Имеются долгоживущие потоки данных для аналитической обработки (окна за период)
- Использование технологии подстройки конвейера обработки для сглаживания пиков
- События должны быть мгновенно приняты в обработку, но лишь часть из них временно сохраняется до завершения обработки

Области применения. Поисковые машины

Кластеризация потоковых данных, включая мониторинг погоды, биржевые торги, анализ веб-сайтов.

Особенности в больших объемах данных и необходимости выполнять группировку событий в реальном времени однократно.

Области применения. Решения "умного города"

Подразумевают наличие умных устройств и сенсоров в общественной инфраструктуре, предназначенной для облегчения жизни людей.

Особенности:

- Потоковые данные от сенсоров могут быть предобработаны ещё до попадания в слой поглощения событий
- Большие объемы данных за короткий интервал времени
- Высокая плотность поступающих событий
- Необходима эластичность потокового приложения в широких диапазонах неоднородной нагрузки

Области применения. Социальные сети

Twitter, LinkedIn, Facebook, Google, Alibaba имеют нагрузку порядка десятков миллионов событий в секунду.

Особенности:

- малая часть потока реально необходима для обработки
- большая часть событий отправляется на долговременное хранение

Использование в поисковых системах

Кластеризация (многомерная) потоковых данных является актуальной проблемой и имеет множество применений, таких как мониторинг погоды, торговля акциями, анализ веб-сайтов. Одним из наиболее часто используемых классов приложений кластеризации являются поисковые системы, которые группируют похожие объекты (например, веб-страницы) в один кластер (например, результаты поиска). Эти приложения характеризуются большим объемом данных, которые необходимо однократно принять и обработать (одно сканирование) в режиме реального времени, после чего записи потоков отбрасываются, а сводная статистика сохраняется для создания кластеров.

Характеристики потока данных

- **Размер данных.** В контексте географически распределённых "туманных"систем, слои приема и хранения, расположенные рядом с источниками данных, могут быть настроены на обработку небольших элементов ("крошечные сообщения в 8–100 байт - десятки байт, "малые 100–1000 байтов - сотни байт). Достоинства же "облачных"систем хранения проявятся лишь для агрегированных потоков, настроенных для средних (несколько килобайт) и больших (более 1 МБ) размеров сообщений.
- **Принципы доступа к данным.** Назначенные операторы потоковой обработки могут использовать интерфейсом ключ-значение, чтобы минимизировать количество обращений к хранилищу данных. Могут обращаться к элементам данных последовательно (scan based), случайным образом или даже с использованием запросов.
- **Модель данных.** Потоки могут быть представлены как простые записи (необязательный ключ, какие-то бинарные объеты в качестве значений атрибутов, необязательная отметка времени), сообщения в очереди, кортежи в хранилище ключ-значение или наборы строк/столбцов в таблице. Характеристики модель данных вместе с размером данных влияют на скорость передачи (на пропускную способность в виде количества записей в секунду).

Характеристики потока данных (продолжение)

- **Компрессия данных.** GZIP, Snappy, LZ4, data encodings, принципы сериализации
- **Размещение данных.** Определяется тем, где физически находятся потоковые данные - в памяти или на флэш-памяти/диске. Например, для запросов архивного потока, выполнении слиния потоков, как и обработки реляционных таблиц, целесообразно хранение по столбцам. Гетерогенные рабочие нагрузки, запросы на обновление, предполагаю гибридную схему хранения строк и столбцов.
  Произвольные типы данных могут опираться на колоночные форматы, такие как Apache ORC, Parquet, CarbonData или использовать сериализаторы Kryo, Avro, которые интегрируются в экосистему Hadoop и обеспечивают статически типизированный интерфейс между производителями и потребителями.

- **Разделения потоков данных.** Потоковые архитектуры могут разбивать потоковые данные на подмножества по временным интервалам, а также использовать други методы разделения: согласованное хеширование, брокер/темы (в Kafka) или сегменты/таблицы (в Druid или Kudu).
- **Унификация потока метаданных.** Потоковые метаданные — это небольшие по объему данные (атрибуты в форме ключ-значение), которые описывают потоки и позволяют находить конкретные данных.
- **Поиск данных.** Возможность поиска по содержимому является большой проблемой, особенно для специализированных систем хранения. Средства приема (ingestion) обычно не поддерживают такую функцию.
- **Маршрутизации сообщений.** Маршрутизация определяет регламент обработки потока сообщений (записей), чтобы обеспечить его обработку и, в конечном итоге, сохранение.
- **Контроль обратного давления.** Противодавление относится к ситуации, когда отдельные компоненты/операторы не обеспечивают скорость, с которой принимаются потоковые данные.

Нефункциональные требования

- **Высокая доступность.** Способность системы в целом работать непрерывно в течение длительного времени, несмотря на частичные сбои (недоступность некоторых узлов в сети или аппаратные сбои).
- **Временное постоянство против долговечности потока.** Временное постоянство относится к способности системы временно сохранять поток событий в памяти или на диске (например, настраиваемый период хранения данных в течение короткого периода времени, после которого данные автоматически удаляются).
- **Масштабируемость.** Способность потокового хранилища обрабатывать растущие объемы данных, сохраняя стабильное время обработки в пиковые моменты, когда объемы данных резко увеличиваются.
- **Задержка обработки и пропускная способность.** Некоторым приложениям требуется доступ к потокам с малой задержкой, в то время как другим требуется высокая пропускная способность, и они могут обеспечивать более высокие задержки.

Основные принципы организации доступа:

- на основе очереди сообщений, которая предоставляет интерфейсы для создания и потребления потоков данных в режиме реального времени;
- на основе платформ общего назначения для приема данных из различных источников (СУБД, REST API, Kafka и других систем).

Apache Kafka представляет собой распределенную потоковую платформу, которая обеспечивает протокол публикации/подписки для гарантированной обработки потоков данных, включая доступ для нескольких потребителей сразу. Это решение с открытым исходным кодом является наиболее распространённым и используется в конвейерах со средствами Apache Spark, Apache Flink, Apache Beam, которые, в свою очередь, обеспечиваю перемещение данных и вычисления. Кластер Kafka состоит из набора узлов-брокеров, которые хранят потоки записей с присвоенной категорией, называемой темами (topics). Каждый поток может быть статически разделен на несколько разделов, что позволяет логически разделить данные потока и распараллелить доступ потребителей.

Apache ActiveMQ предоставляет брокеры сообщений, которые реализуют спецификацию Java Message Service (JMS) и обеспечивают высокую доступность, масштабируемость, надежность и безопасность для корпоративного обмена сообщениями. ActiveMQ предоставляет множество интерфейсов, включая поддержку таких протоколов, как HTTP/S, SSL, STOMP, TCP, UDP, XMPP и многоадресная рассылка IP. Существует два режима обмена сообщениями в JMS:

- PTP (точка-точка) использует очереди назначения, через которые сообщения отправляются и принимаются синхронно или асинхронно, причем каждое сообщение доставляется потребителю ровно один раз;
- pub/sub (публикация/подписка) предполагает, что производители отправляют сообщения в рамках назначенной темы (topic), а подписчики регистрируются синхронно или асинхронно для получения сообщений. Тема может быть настроена как сохраняемая подписка, гарантирующая, что в случае отключения подписчиков, сообщения будут доставлены после их появления.

PTP не гарантирует сохранности сообщения. ActiveMQ поддерживает два типа доставки сообщений: постоянное сообщение регистрируется в постоянном хранилище, в то время как доставка непостоянного сообщения не гарантируется.

RabbitMQ реализует брокер сообщений с поддержкой протокола Advanced Message Queuing Protocol (AMQP). Написан на языке Erlang. Модель AMQP основана на обменах, что предполагает наличие агента маршрутизации сообщений, связывающего разные очереди между собой. Вместо того, чтобы публиковать сообщения непосредственно в очереди, производитель данных отправляет сообщения в точку обмена. Возможно связывание нескольких очередей, возможна назначение ключа конкретным очередям. А каждое сообщение имеет ключ маршрутизации для, собственно, выполненния их правильной маршрутизации в нужную очередь. Поддерживается несколько режимов обмена:

- прямой обмен доставляет сообщения в очередь, если ключ привязки точно совпадает с ключом маршрутизации сообщения, обмен темами — это маршрутизация сообщений на основе ключей маршрутизации, соответствующих шаблону маршрутизации, заданному привязкой очереди,
- широковещательный обмен просто копирует и направляет сообщение во все связанные очереди, игнорируя ключи маршрутизации или шаблоны привязки. Потребители могут работать в режиме получения сообщений по приходу (push API) или извлечения сообщений из очередей по запросу (pull API).

- **ZeroMQ** представляет собой встраиваемое решение (в виде библиотеки) для асинхронного обмена сообщениями и не требует специального брокера сообщений. ZeroMQ обеспечивает работу прямых и разветвленных соединений с потребителями и реализует различные шаблоны обмена сообщениями, такие как запрос/ответ, публикация/подписка, push/pull.
- **HornetQ** (устарел) реализует спецификацию JMS. HornetQ может работать как автономный сервер, встраиваемый контейнер и ввиде кластера, включая репликацию серверов и отказоустойчивый клиент. HornetQ предоставляет отправку и получения очень больших >1 ГБ сообщений и обеспечивает поддержку контроля обратного давления. С 2015 года кодовая база передана в проект ActiveMQ архитектура Artemis.
- **ActiveMQ Artemis**. Предоставляет брокеры сообщений, поддерживающих спецификации JMS 1.1 и 2.0 с полной реализацией, включая JNDI. Язык реализации - Java. Архитектура "Artemis в отличии от ActiveMQ 5 “Classic”, является не блокирующей и ориентирована на построение приложений с логикой, управляемой сообщениями. Возможна интеграция с другими протоколами типа AMQP и Streaming Text Oriented Messaging Protocol (STOMP).

DistributedLog — сервис регистрации операций с георепликированием. Реализован в двухуровневой архитектуре, которая позволяет независимо масштабировать операции чтения и записи. DistributedLog использовался для построения различных систем обмена сообщениями, включая транзакционную поддержку. Поток сообщений статически разделяется на фиксированное количество разделов, каждый из которых поддерживается журналом транзакций; сегменты журнала распределены по нескольким узлам, управляемым Bookkeeper, который представляет собой масштабируемую, отказоустойчивую службу хранения, оптимизированную для рабочих нагрузок в реальном времени. В DistributedLog в конкретный момент времени существует только один активный модуль записи журнала. Потребитель сообщений начинает читать записи с определенной позиции (смещения), пока не дойдет до хвоста журнала. Поле этого, потребитель ожидает уведомления о новых сегментах или записях журнала.

С 2017-го года является частью проекта Apache BookKeeper.

Apache Pulsar представляет собой систему обмена сообщениями, разработанную на основе Bookkeeper, с двухуровневой архитектурой, состоящей из уровня обслуживания без сохранения состояния и уровня сохранения состояния. Изначально разработан в компании Yahoo!. В сравнении с DistributedLog, операции чтения и записи не могут масштабироваться независимо друг от друга (первый уровень используется как для чтения, так и для записи). При этом, клиенты Pulsar не взаимодействуют с Bookkeeper напрямую. Pulsar объединяет модели очередей и тем, обеспечивая режимы эксклюзивной, разделяемой и отказоустойчивой подписки. Pulsar отслеживает прочитанные потребителем записи, имея возможность удалять их только после подтверждения потребителями.

Pravega — еще одна система хранения потоков с открытым исходным кодом, построенная на основе Bookkeeper. Pravega разделяет поток на фиксированное количество разделов, называемых сегментами, имея одноуровневую архитектуру брокеров сообщений. Pravega обеспечивает автоматическое масштабирование количества сегментов (разделов) в потоке, а на за счёт мониторинга входной нагрузки (размера или количества событий) может объединять несколько сегментов в один или создавать новые. Производители сообщений могут разделять поток только по ключу записи.

Apache Flume — это распределенный сервис для эффективного сбора, агрегирования и перемещения больших объемов журналов операций. Типовые области применения - сбор данных о сетевом трафике, сообщений социальных сетей, сообщений электронной почты из различных источников в централизованные хранилища данных. Архитектурно реализован в виде агентов Flume, каждый из которых представляет собой процесс JVM, имеющего компоненты источника, канала передачи и приемника данных. Агент может передавать данные следующему агенту. Надежность передачи данных обеспечивается за счёт записи в файл временногр хранения событий (каналы могут поддерживаться очередью в памяти, но события будут потеряны в случае сбоя). В конечном итоге, данные доставляются потребителю и могут быть отправлены во внешний репозиторий, такой как HDFS.

Apache Gobblin представляет собой среду интеграции данных, ориентированную на пакетный режим обработки, изначально базирующийся на Hadoop HDFS. Apache Gobblin создает цепочки обработки данных. Каждое задание может включать в себя источник данных, экстрактор, преобразователь, средства проверки качества, средства записи в долговременную память и средства публикации. Источниками данных могут быть реляционные хранилища, потоковые системы, REST, файловые системы, темы Apache Kafka. Gobblin поддерживает абстракцию на уровне записей. В планах добавить поддержку уровня файлов. Механизм обратного давления основан на двух методах: слиянии элементов обработки для уменьшения количества небольших файлов, публикуемых в HDFS, и балансировке нагрузки элементов обработки в контейнерах.

- **Elasticsearch** представляет собой распределенную поисковую систему в реальном времени, предназначенную для приема и хранения документов JSON, основанную на Logstash для приема и преобразования данных на лету. Использует Apache Lucene для индексации и хранения файлов и метаданных на диске в настраиваемых каталогах данных. Elasticsearch интегрируется с Hadoop, с помощью которого пользователи могут создавать динамические встроенные поисковые приложения для обслуживания данных на основе HDFS или выполнять аналитику с малой задержкой, используя полнотекстовые геопространственные запросы и агрегации. Elasticsearch имеет возможность индексировать данные в полях меток времени и предоставляет примитивы для хранения и запроса данных временных рядов.
- **Apache Sqoop** (переведён в устаревшие проекты) — это инструмент, предназначенный для эффективной передачи больших объемов данных между HDFS и слабо структурированными хранилищами данных, например HBase, Cassandra, реляционные базы данных.

Классификация потоковых систем хранения:

- хранилища ключ-значение (с операциями добавить/получить по ключу);
- специализированные хранилища для потоковых систем (графовые потоковые приложений);
- базы данных для временных рядов (ориентированы на IoT-приложений);
- структурированные хранилища (поддержка запросов диапазона);
- поколоночные хранилища поверх Hadoop HDFS.

Redis реализует хранилище данных в памяти. Может использоваться в качестве базы данных ключ-значение, кэша и брокера сообщений. Redis поддерживает множество типов данных для хранения, например строки, списки, хэши, можества, сортированные множества, битовые массивы, а также обеспечивает поиск по геопространственным индексам. Redis реализует парадигму обмена сообщениями pub-sub и группирует сообщения в каналы с подписчиками, проявляющими интерес к одному или нескольким каналам. Redis обеспечивает постоянное хранение, делая сбрасывая данные из оперативной памяти на диск, но не обеспечивает строгой согласованности.

RAMCloud — это экспериментальное хранилище ключ-значение в памяти, предназначенное для чтения и записи с малой задержкой за счет использования высокопроизводительных сетей, подобных Infiniband. Надежность и доступность гарантируются за счет репликации данных на удаленные диски серверов, работающих от источников бесперебойного питания. К особенностям RAMCloud относится быстрое восстановление после сбоев, эффективное использование памяти и строгую согласованность данных. RAMCloud имеет вторичные индексы, которые позволяют добиться высокой доступности за счет распределения индексов независимо от объектов, на которые они указывают.

MICA представляет собой экспериментальное масштабируемое хранилище ключ-значение в памяти, обеспечивающее стабильно высокую пропускную способность при различных смешанных рабочих нагрузках чтения и записи при использовании многоядерной вычислительной системы общего назначения с общей памятью. MICA обеспечивает параллельный доступ к данным, разбитым по разделам, и использует упрощенный сетевой стек, который обходит ядро операционной системы (непосредственно взаимодействует с сетевыми адаптерами). Программы, использующие стандартные сокеты ввода-вывода, оптимизированные для типовых задач коммуникации, имеют большие накладные расходы сетевого стека как на уровне ядра, так и на пользовательском уровне. MICA предназначена для достижения высокой пропускной способности одного узла, стабильной производительности при разных рабочих нагрузках, обеспечивает низкую сквозную задержку на оборудовании общего назначения при условии обработки небольших ключей переменной длины.

HyperDex представляет собой исследовательский проект, завершенный в 2016-м году. Реализует распределенное хранилище ключ-значение, которое предоставляет возможность поиска по вторичным атрибутам в дополнение к поиску объектов по его первичному ключу. HyperDex организует свои данные с помощью метода, называемого хешированием в гиперпространстве (многомерном евклидовом пространстве), который сопоставляет объекты хранения с серверами (принимая во внимание вторичные атрибуты), что обеспечивает эффективное выполнение операций вставки и поиска. HyperDex обеспечивает строгую согласованность и отказоустойчивость для одновременных обновлений, используя метод, в котором операции обновления объектов упорядочиваются в последовательность, зависящую от из значений, определяющих координаты гиперпространства объекта. HyperDex обеспечивает согласованность как на уровне ключей, так и поиска значений, чтобы гарантировать, что поиск вернет именно те объекты, которые были выявлены в процессе.

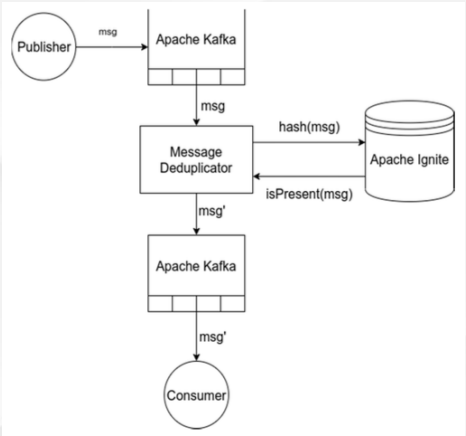
Другие способы хранения потоковых данных

Иные средства промежуточного хранения, обслуживающие какие-либо специальные задачи:

- Memcached
- RocksDB (встраиваемое хранилище ключ-значение)
- LMDB (Lightning Memory-Mapped Database)
- Ignite Persistence
- ...

M. M. Rovnyagin, V. K. Kozlov, R. A. Mitenkov,

A. D. Gukov, and A. A. Yakovlev. Caching and storage optimizations for big data streaming systems.
In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 468–471, 2020



Потоковые архитектуры развиваются как часть комплексных систем обработки информации, поэтому должен быть универсальный механизм обмена данными.

Распределенная файловая система Hadoop **HDFS** обеспечивает масштабируемое и надежное хранилище данных (сходна с GoogleFS) и признана стандартом де-факто для хранения больших данных для аналитики.

Несмотря на то, что HDFS не предназначена для потоков, многие потоковые фреймворки (Apache Spark, Apache Flink и пр.) используют её как источник или сток для данных. Иногда, для для хранения контрольных точек).

Ограничение HDFS – сервер метаданных (NameNode), который является единой точкой отказа и причиной ограниченной масштабируемости.

HDFS не поддерживает запись в произвольную позицию.

HDFS плохо справляется с управлением большим количеством небольших файлов.

DXRAM — исследовательский проект Heinrich-Heine-University Düsseldorf. Реализует хранилище ключ-значение, которое всегда хранит данные в оперативной памяти и предназначено для миллиардов небольших объектов данных (16–128 байт), необходимых графическим приложениям. DXRAM реплицирует свои данные, используя асинхронное ведение журнала операций на удаленных дисках. Кроме того, разработан метод ведения журнала с учетом специфики SSD, чтобы оптимизировать пропускную способность чтения и записи для небольших кортежей «ключ-значение».

Hyperion – исследовательский проект University of Massachusetts, представляет собой систему для архивирования, индексации и оперативного поиска больших потоков данных из сетевых заголовков. Архивирование данных для систем сетевого мониторинга осложняется скоростью их поступления и необходимостью индексирования на лету для поддержки ретроспективных запросов. Hyperion обеспечивает выполнение запросов по диапазону и рангу. Новые записи добавляются последовательно и не изменяются, что позволяет уменьшить накладные расходы на поиск на диске и повысить пропускную способность системы. Оптимизированная для записи потоков файловая система обеспечивает гранулярность на уровне записи, а не файла.

Apache Druid — это распределенное хранилище данных с открытым исходным кодом с поколоночным хранением. Предназначено для аналитики наборов больших данных в режиме реального времени. Druid решает задачу приема данных и немедленного их предоставления для обработки, что отличает его от Hadoop HDFS, где хоть и обеспечена высокая доступность, но производительность снижается при высокой параллельной нагрузке.

Кластер Druid состоит из специализированных узлов:

- узлы реального времени, которые поддерживают индекс в памяти для всех входящих событий и регулярно сохраняют его на диск. Также принимают поток данных и запросы к потокам;
- "исторические узлы"обеспечивают доступ к неизменяемым блокам данных, которые создаются узлами реального времени;
- узлы-брокеры работают как маршрутизаторы запросов от узлов реального времени к историческим узлам (метаданные публикуются в ZooKeeper);
- узлы-координаторы, отвечающие за управление данными и распределение исторических узлов (загрузка новых данных, удаление старых , репликация, балансировка нагрузки).

Модель данных Druid основана на элементах данных с отметками времени (например, журналы сетевых событий). Druid требует столбец меток времени для разделения данных, что позволяет обслуживать запросы в заданных диапазонах времени с малой задержкой.

- **TimescaleDB** — это надстройка над PostgreSQL для расширения и масштабирования SQL для временных рядов. Основная задача состоит в том, чтобы поддерживать высокую скорость приема данных временных рядов. Для этого делаются следующие допущения: данные временных рядов неизменяемы, а записи добавляются в последние временные интервалы. Рабочие нагрузки естественным образом распределяются по времени и пространству.
- **InfluxDB** — это платформа, предназначенная для срочных данных (например, метрик, событий). InfluxDB хранит данные на диске (структурированное по времени дерево слияния) и предоставляет SQL-подобный язык запросов на основе JSON. В тесте, который измеряет производительность приема данных в секунду, требования к хранилищу на диске (сжатие) и среднее время ответа на запрос (миллисекунды), InfluxDB превосходит Elasticsearch до 10 раз. Версия с открытым исходным кодом не поддерживает кластеризацию, но для обеспечения высокой доступности и горизонтальной масштабируемости доступно коммерческое решение.
- **RiakTS** — это корпоративная СУБД для временных рядов класса nosql с открытым исходным кодом, оптимизированная для IoT и, собственно, временных рядов.
- **OpenTSDB** — масштабируемая СУБД для временных рядов с открытым исходным кодом, работающая на Hadoop и HBase.

Apache Kudu — это поколоночное хранилище данных, которое интегрируется с Apache Impala (SQL query engine), HDFS и HBase. Его можно рассматривать как альтернативу Avro/Parquet поверх HDFS, которые не могут эффективно обрабатывать случайные операции чтения/записи. Также является альтернативой хранилищ слабо структурированных данных типа HBase и Cassandra. В отличии от них, обеспечивает доступ с малой задержкой на чтение и запись (6 ms для 99%). А также пригоден для последовательного чтения, необходимого приложениям машинного обучения или SQL.

Кластер Kudu представляет собой набор таблиц: каждая таблица имеет четко определенную схему и состоит из конечного числа столбцов; каждый столбец определяется типом и определяет первичный ключ, но не вторичные индексы. Kudu предоставляет быстрое сканирование столбцов (сравнимое с Parquet, ORC) и произвольные обновления с малой задержкой.

Apache HBase — это хранилище реального времени, которое поддерживает поиск и изменение индексированных записей по ключу (произвольный доступ для чтения/записи к большим данным в реальном времени). HBase — это распределенная, версионная, нереляционная СУБД, созданная по образцу Google BigTable.

Apache Cassandra — это распределенная система хранения больших объемов структурированных данных, разбросанных по множеству серверов общего назначения. Cassandra обеспечивает высокую доступность (без единой точки отказа) и надежность посредством репликации. Cassandra не поддерживает полную реляционную модель данных. Таблица представляет собой распределенную многомерную карту, индексированную по ключу. API содержит три метода:

- вставка (таблица, ключ, rowMutation)
- получение (таблица, ключ, имя столбца)
- удаление (таблица, ключ, имя столбца)

Реализовн язык запросов Cassandra (CQL). Cassandra может масштабироваться постепенно, динамически разделяя данные по набору узлов. Для этого используется согласованное хеширование.

- **Apache CarbonData** — это полностью индексированное поколоночное хранилище данных для быстрой обработки аналитических запросов к потокам больших данных. По сравнению с традиционными форматами столбцов, в CarbonData столбцы в каждой группе строк сортируются независимо от других столбцов. CarbonData подерживает различные моделей доступа к данным. Может использовать последовательный, произвольный доступ и выполнять аналитическую обработку (запросы с малой задержкой к свежим данным). Предоставляет интерфейсы для интеграции со Spark SQL с поддержкой приема больших объемов данных.
- **Apache Parquet** — это колоночный формат хранения поверх HDFS, обслуживающий сложные вложенные структуры данных. Parquet реализует методы, описанные в проекте Google Dremel, и создан для поддержки эффективных схем сжатия и кодирования данных. Тип сжатия в Parquet по умолчанию — Snappy. Обеспечена интеграция в Apache Spark SQL.
- **Apache Orc** — это поколоночный интерфейс для Hadoop, который включает поддержку транзакций ACID и изоляцию моментальных снимков. Сжатие Orc по умолчанию — ZLIB. Orc хорошо интегрирован с Apache Hive.
- **ClickHouse** - поколоночная СУБД для сверх больших объемов данных.

Отсутствующие функции хранилищ потоковых данных

- **Языки запросов** или SQL в потоках появились как удобная абстракция для обработки входящих данных. Однако они выходят за рамки моделей потоков, основанных на времени или кортежах. Семантики потокового SQL достаточно сложная, при этом не обеспечивается оптимизация доступа, поскольку вычисление агрегатных функций в потоках требует различных оконных конструкций (скользящих, перекрывающиеся окна и пр.). Это требуют низкий уровень доступа к данным.
- **Детектор режима доступа.** Средства приёма и хранения данных, при наличии детектора режима доступа и последующей динамической адаптации к данным, могли бы оптимизировать режим доступа. Например отличать доступ для каждой записи/кортежу, групповые запросы с массовым чтением/записью или длинные последовательные операции.
- **Потоковое машинное обучение.** Применение машинного обучения к историческим данным и данным реального времени – это актуальная тема исследований. Например, имеются проблемы кластеризацию потоковых данных в сравнении с архивными данными.
- **Потоковые графы.** Обработка массивных графов в модели потоков данных требует поддержки со стороны системы хранения потоков для материализации динамических сложных структур данных и реализации пользовательских методов разделения.

- **Работа с окнами.** Окна – основной примитив потоковой обработки. Потоковые фреймворки имеют механизмы отказоустойчивого хранения состояния окна. Но это может быть реализовано на уровне приема/хранения данных. Прочие потоковые операции также требуют хранение локального состояния, которое обычно реализуется созданием контрольных точек в локальных или распределенных файловых системах, но не на уровне хранилищ.
- **Вычисление агрегатных функций в хранилище.** Позволяет уменьшить объемы данных при перемещени по сети и минимизировать расходы на сериализацию и десериализацию. Предварительная обработка и агрегация данных должны выполняться там, где хранятся данные. Современные хранилища размещаются на узлах с большым объемом памяти.
- **Поддержка географически распределенной репликации.** Репликация для "Fog"и гибридных "fog – cloud"систем с несколькими центрами обработки данных необходима для обеспечения высокой доступность услуг, включая отказоустойчивость в масштабах всего региона и обеспечение локальности чтения.

Направления будущего развития систем приёма и хранения

- Гибкость: способность адаптироваться под новые требования
- Специализированные архитектуры потокового хранилища
- Интеграция в экосистему больших данных: место HDFS
- Улучшенная поддержка fog/edge вычислений

[1] O. Marcu. *KerA: A Unified Ingestion and Storage System for Scalable Big Data Processing. (KerA : un système unifié d'ingestion et de stockage pour le traitement efficace du Big Data)*. PhD thesis, INSA Rennes, France, 2018.

[2] O.-C. Marcu, A. Costan, G. Antoniu, M. Pérez-Hernández, R. Tudoran, S. Bortoli, and B. Nicolae. *Storage and Ingestion Systems in Support of Stream Processing: Survey*. PhD thesis, INRIA Rennes-Bretagne Atlantique and University of Rennes 1, France, 2018.

[3] M. M. Rovnyagin, V. K. Kozlov, R. A. Mitenkov, A. D. Gukov, and A. A. Yakovlev. Caching and storage optimizations for big data streaming systems. In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 468–471, 2020.