Modern Storages and Data Warehousing
Week 11 - Streaming

Попов Илья, i.popov@hse.ru

1 - Homework #4

2 - Homework Q&A

Recap прошлых занятий

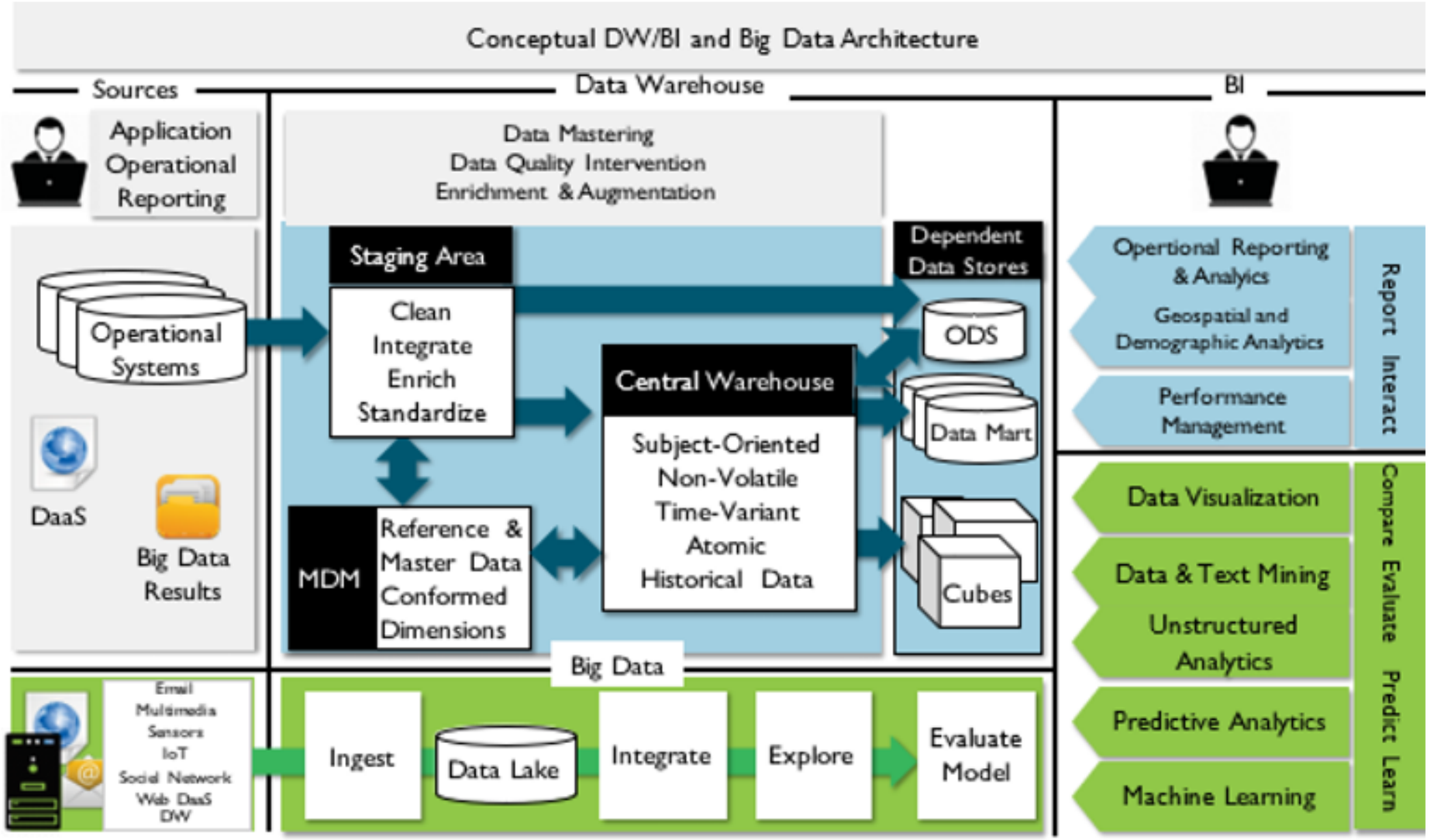


Figure 5: Date Warehouse Concept

3 - Streaming

Мотивация

〉 На курсе мы с вами научились вертеть данные самыми разными способами

〉 Хотим научиться в потоковую обработку данных

Мотивация

〉 На курсе мы с вами научились вертеть данные самыми разными способами
〉 Хотим научиться в потоковую обработку данных

С помощью потоковой обработки мы можем:
〉 Уменьшить задержку в поставке данных
〉 Реализовывать неочевидные сценарии (например, дедубликатор)
〉 Вычислять на потоковых данных метрики и строить над ними систему мониторинга

Что мы с вами уже знаем

〉 Брокеры сообщений

Что мы с вами уже знаем

- Брокеры сообщений

- Из топиков можно прочитать сообщения:
  - Прочитать и обработать с помощью консьюмера
  - Прочитать и сложить для дальнейшей обработки

Демо 1

Что мы с вами уже знаем

- Брокеры сообщений

- Из топиков можно прочитать сообщения:
  - Прочитать и обработать с помощью консьюмера
  - Прочитать и сложить для дальнейшей обработки

- Вопрос:

Что мы с вами уже знаем

- Брокеры сообщений

- Из топиков можно прочитать сообщения:
  - Прочитать и обработать с помощью консьюмера
  - Прочитать и сложить для дальнейшей обработки

- Вопрос:
  - Почему бы не сделать так же, как мы уже видели на примере MR, и придумать фреймворк, который поможет ~~легко~~ легче работать с потоками

Что мы с вами уже знаем

- Брокеры сообщений

- Из топиков можно прочитать сообщения:
  - Прочитать и обработать с помощью консьюмера
  - Прочитать и сложить для дальнейшей обработки

- Вопрос:
  - Почему бы не сделать так же, как мы уже видели на примере MR, и придумать фреймворк, который поможет ~~легко~~ легче работать с потоками
  - И желательно - не ходить далеко и использовать знакомый стек

Spark Streaming

- Привет, знакомый нам Spark!
- Разработка уходит корнями в Berkley
- Относится к near-realtime системам, т.е. с относительно небольшой задержкой

Spark streaming:

- оперирует понятием micro-batch
- Discretized Streams (DStreams):
  - появился в Spark 0.7.0
  - RDD-based batch, на каждый micro-batch создаётся RDD
  - считается устаревшим

Spark Streaming

Structure streaming:

〉Появился в Spark 2.0 (stable в 2.2),

〉DataFrame API: позволяет смешивать классические batch-обработки и обработки потоковой информации, используя один и тот же API

〉Поддержка watermarks (метки для использования данных прошлых micro-batch)

〉Оптимизатор Catalyst

〉Постоянно развивается

Главное отличие Structure streaming от Discretized Streams заключается в том, что полученную информацию представляем как большую «бесконечную» таблицу, которую мы можем обрабатывать.

Spark Streaming

Fault tolerance:

〉checkpoint: информация о последних обработанных offset per partition per topic, которая складывается в папку, указанную на момент запуска приложения;

〉несколько политик для trigger: определяют время обработки потоковых данных, будет ли запрос выполняться как micro-batch запрос с фиксированным интервалом batch обработки или как запрос непрерывной обработки;

〉динамические метрики: можем получать информацию о том, что сейчас происходит в поточной обработке.

Structure streaming source

Есть источник и приёмник и между ними выстраиваем логику взаимодействия.

По типу представления источники делятся на категории:

〉 File (файловый источник): text, CSV, JSON, ORC, Parquet,

〉 Kafka,

〉 Socket – чтение данных из сокета (обычно для отладки), подключение по IP-адресу к порту,

〉 Rate – для генерации данных (тестирование + benchmarking), генерирует случайным образом нужное количество сообщений в секунду.

Structure streaming sink

По типу представления приёмники делятся на категории:

〉 File (файловый приёмник),

〉 Kafka,

〉 Foreach (foreach & foreachBatch) – позволяет применять операции каждому row / batch. Также позволяет несколько раз переиспользовать данные и писать в нескольких output (что нельзя делать для других типов sink),

〉 Console – для отладки и тестирования, вывод информации в консоль,

〉 Memory – хранит данные в памяти (table_name == query_id) – обычно для отладки и демо-примеров.

Structure streaming triggers

Trigger – правило срабатывания micro-batch-а.

〉 **Unspecified** (default) – запуск micro-batch по готовности:

spark.streaming.receiver.maxRate

spark.streaming.kafka.maxRatePerPartition

〉 **Fixed interval** – запуск через равные промежутки времени:

предыдущий успел выполниться – ждёт конца интервала,

предыдущий не успел выполниться – ждёт окончания и сразу запускается,

нет данных – не запускается.

〉 **One-time** – запускается один раз, получает все доступные данные и обрабатывает, после чего останавливается (для небольшого количества информации).

〉 **Continuous with fixed checkpoint interval** – экспериментальный – позволяет обрабатывать данные с малой задержкой (~1 ms)

Демо 2

KSQL

- Привет, знакомая нам Kafka!
- Система, позволяющая писать SQL-like запросы на потоках данных
- Работает на Kafka Streams — Java API, которое позволяет перекладывать данные из топика в топик, по пути совершая различные преобразования этих данных.
- Главные абстракции:

  Потоки — неизменяемые последовательности событий, доступные только для добавления. Они полезны для представления ряда исторических фактов.

  Таблицы — это изменяемые коллекции событий. Они позволяют представлять последнюю версию каждого значения для каждого ключа.

KSQL

- **Запросы** (queries) могут быть 2-х видов – push и pull.
- **Push-запросы** позволяют подписаться на результат по мере его изменения в реальном времени. При поступлении новых событий push-запросы производят уточнения, чтобы быстро реагировать на новую информацию. Push-запросы идеально подходят для асинхронных потоков приложений.
- **Pull-запросы** позволяют получить текущее состояние материализованного представления, которые постепенно обновляются по мере поступления новых событий. Поэтому pull-запросы выполняются с предсказуемо низкой задержкой и отлично подходят для потоков типа «запрос/ответ» (request/response).

Демо 3

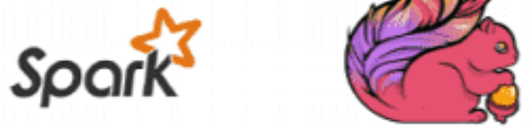
Flink

- Еще один зверь из Apache Foundation
- Разработка уходит корнями в TU University (Berlin)
- Под капотом - Java, хотя есть и Table API / Python SDK
- Де-факто - текущий стандарт индустрии

Flink vs Spark

Batch Comparison

API	high-level	high-level
Data Transfer	batch	pipelined & batch
Memory Management	JVM-managed	Active managed
Iterations	in-memory cached	streamed
Fault tolerance	task level	job level
Good at	data exploration	heavy backend & iterative jobs
Libraries	built-in & external	evolving built-in & external



Streaming Comparison

Streaming	mini batches	“true”
API	high-level	high-level
Fault tolerance	RDD-based (lineage)	coarse checkpointing
State	external	internal
Exactly once	exactly once	exactly once
Windowing	restricted	flexible
Latency	medium	low
Throughput	high	high

Плюсы и минусы стриминга

Плюсы:

〉 Меньше SLA на поставку данных

〉 Можно делать сложные сценарии на потоках данных

〉 Инкрементальность из коробки

Минусы:

〉 Сложно поддерживать

〉 Выделение дополнительных ресурсов, поддержка 9999

〉 Из DE превращаемся в разработчиков-enjoyer’ов

Надежность процессов

9999

〉 Тиры надежности (они же “девятки”) - мера отказоучтойчивости высшего сервиса

〉 Обычно процессы объединяют в классы надежности (они же “тиры”). Где-то говорят Tier A -> Tier D, где-то - Tier 0 -> Tier 3 - смысл от этого не меняется.

〉 Чем выше класс - тем жестче требования к сервису.

〉 Из требований могут быть: аптайм, мониторинги, алерты (втч инциденты), дежурства, документация, требования к релейному процессу и код ревью и т.д.

Задача на System Design

Проектируем дедубликор в 9999

〉Дедубликатор - инструмент для удаления дублей в потоке данных

〉Есть входной поток - например, топик в Kafka

Проектируем дедубликор в 9999

〉 Дедубликатор - инструмент для удаления дублей в потоке данных

〉 Есть входной поток - например, топик в Kafka

〉 Окно 30 сек
〉 RPS ~100