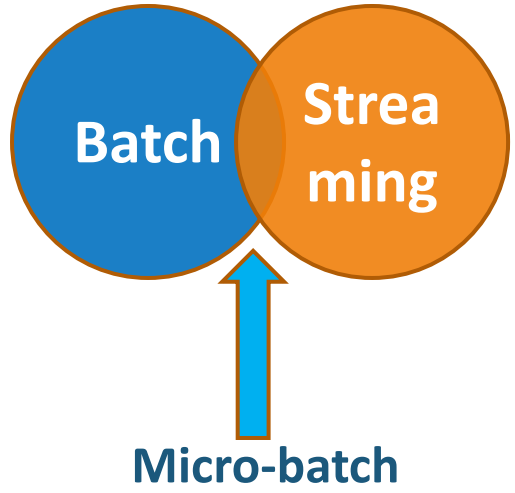
Spark streaming

МОДУЛЬ 9

ШКОЛА БОЛЬШИХ ДАННЫХ

Streaming vs Batch



- **“batch”** - обработка всех данных
- **“streaming”** - обработка постоянно поступающих данных
- постоянно поступать могут как структурированные, так и не структурированные данные
- желательно обрабатывать единообразно, минимальным количеством инструментов

Примеры потоковой обработки

- нотификация и алерты
  - ✓ подбор параметров для минимизации **страховой премии**
- отчетность в реальном времени
  - ✓ **market value** инвестиционного портфеля
- принятие решений в реальном времени
  - ✓ автоматическая **блокировка fraud** транзакции
- ML в реальном времени
  - ✓ постоянное **переобучение** модели на потоковых данных

Пример: подбор параметров страхового тарифа

- **онлайн** покупка страхового продукта
  - ✓ страхователь общается с сайтом
  - ✓ бОльшую часть предоставленной информации проверить невозможно
- **тариф** зависит в том числе от
  - ✓ данных страхователя
  - ✓ данных допущенных к управлению
  - ✓ данных ТС
- схема **мошенничества**
  - ✓ заполняем форму - получаем тариф
  - ✓ меняем данные в форме - получаем другой тариф
  - ✓ подбором параметров минимизируем тариф
- **мошенничество** - предоставление ложной информации
  - ✓ VIN (с минимальной ошибкой)
  - ✓ персональные данные (аналогично)
  - ✓ искаженная адресная и другая информация
- как **бороться**
  - ✓ мониторить "подборы"
  - ✓ пресекать на N-ном шаге ("обратитесь к андеррайтеру")
- **проблемы**
  - ✓ алгоритмическая сложность
  - ✓ техническая сложность (хранение состояния, время отклика)
- **решение**: закрытие очевидных "дыр» постфактум

Сложности потоковой обработки

- “event time”: обработка **“out of order”** событий
- Поддержка **состояния** (в оперативной памяти)
- работа с высокой нагрузкой (большой **“thoughput”**)
- обработка **“exactly once”**
- реакция на события "в реальном времени" (**минимизация задержек**)
- транзакционная запись в системы хранения
- обновление логики системы в реальном времени

Continous vs micro batch

- обработка каждой записи vs декларативное API
  - ✓ Apache Storm: one-record-at-a-time
  - ✓ Apache Spark: декларативное API
- непрерывная обработка vs micro-batch
  - ✓ Apache Nifi: непрерывная обработка
  - ✓ Apache Spark: micro-batch (в будущем возможна непрерывная обработка)

Обработка потоковых данных

(на примере “Apache NiFi” и задач инженерии данных)

Настраиваем граф обработки, который

- ✓ периодически **читает** данные (**файлы**, **таблицы**)
- ✓ **обрабатывает** каждый элемент
- ✓ **собирает** в **"batch"** (где-то временно храня)
- ✓ **записывает** batch в хранилище (например, **Hive**)

Минусы подхода

Все это мы делаем

- ✓ **другим** инструментом
- ✓ на каком-то **другом** оборудовании (не в кластере)
- ✓ **каким-то** образом обеспечиваем мониторинг
- ✓ **каким-то** образом обеспечиваем `lineage`

Spark Streaming APIs

- DStreams (2012)
  - ✓ RDD (pyspark.streaming)
  - ✓ минусы
    - ➢ работа на **“низком уровне”** (просто с объектами)
    - ➢ базируется на **“processing time”** (не “event time”)
    - ➢ строго **“micro-batch”**
- Structured Streaming
  - ✓ DataFrame (**pyspark.sql.streaming**)
  - ✓ единый подход (**API**) - единый код (**batch + streaming**)

Spark Structured Streaming

Что предлагает **Spark Structured Streaming**

- единую абстракцию (**“dataframe”**)
- простые механизмы получения потоковых данных (например, чтение из файлов)
- новое действие(я) (**“action”**): трансформация запускается при обновлении данных
- **итого**: возможность работы с актуальными данными (**“batch + streaming”**)

Основные абстракции

- **“streaming dataframe”**: стриминговый датафрейм
  - ✓ очень похож на просто датафрейм
  - ✓ создается при помощи **“pyspark.sql.streaming.DataStreamReader”**
  - ✓ трансформации практически не изменились
  - ✓ действия - другие (**нет “show()”** - поток...)
  - ✓ новое действие **“writeStream()”**
- **“streaming query”**: механизм управления потоками
  - ✓ **“stop()”** для остановки потока
- **“source”** – **“sink”** парадигма
  - ✓ читаем из источника
  - ✓ пишем в **“sink”**
  - ✓ один “sink” - один “source”

Как это работает

(на примере файлов)

- **“source”**: создаем датафрейм из файла (**“readStream”**)
- настраиваем необходимые преобразования (как обычно – **“dataframe”**)
- настраиваем параметры стриминга (**интервал**)
- **“sink”**: определяем - куда записывать результаты (**“writeStream”**)
- запускаем стриминг (**“start”**)

Обеспечение надежности

- потоковые данные не всегда можно "перезапросить"
- потоковая обработка работает **24*7**
- надежность важна
  - ✓ минимизация вероятности потери данных
  - ✓ минимизация сложности восстановления обработки
  - ✓ минимизация downtime

Checkpointing в Spark

- способ надежного сохранения "промежуточных" результатов
- два вида **checkpoint**
  - ✓ dataframe
  - ✓ streaming
- **“streaming checkpoint”**
  - ✓ задаем директорию (`**.option("checkpointLocation", "path/to/HDFS/dir")**`)
  - ✓ в этой директории хранится **метаинформация**
  - ✓ при запуске после сбоя обработка будет продолжена (**без потери данных**)

Вопросы?