Интеграция Spark ML Pipeline в высоконагруженный low-latency сервис рекомендаций

Малов Илья

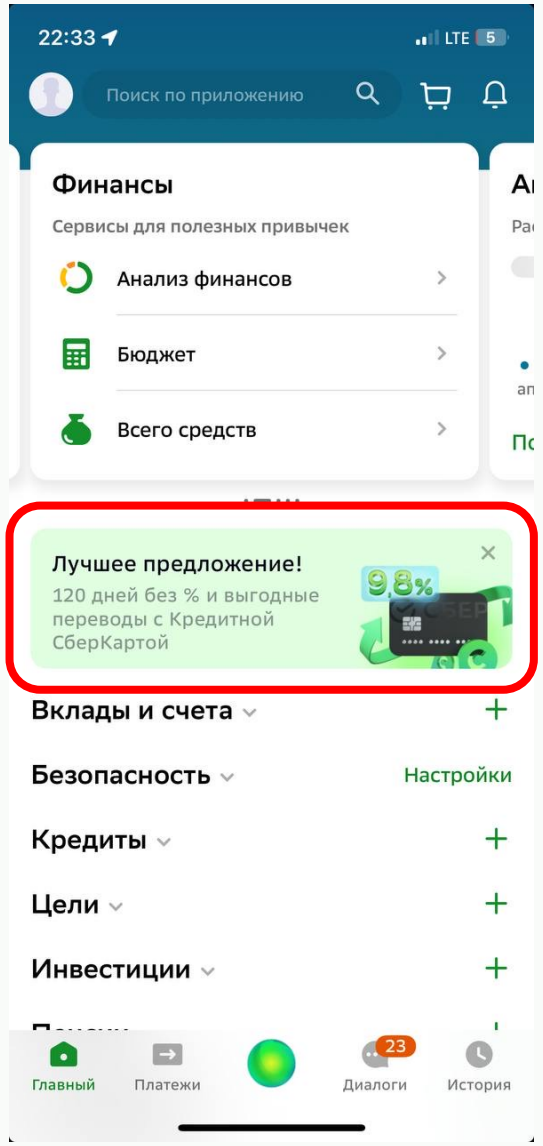
Сбер

О чём будем говорить

- Рекомендации для СБОЛа
- Модель в мире больших данных
- Сервис в мире хайлоада
- Как перейти из мира больших данных в хайлоад

Ранжирование предложений для СБОЛа

- Из списка возможных предложений нужно подобрать наиболее релевантное
- Наиболее релевантное предложение определяется с помощью ml-модели
- Всё это работает в виде сервиса



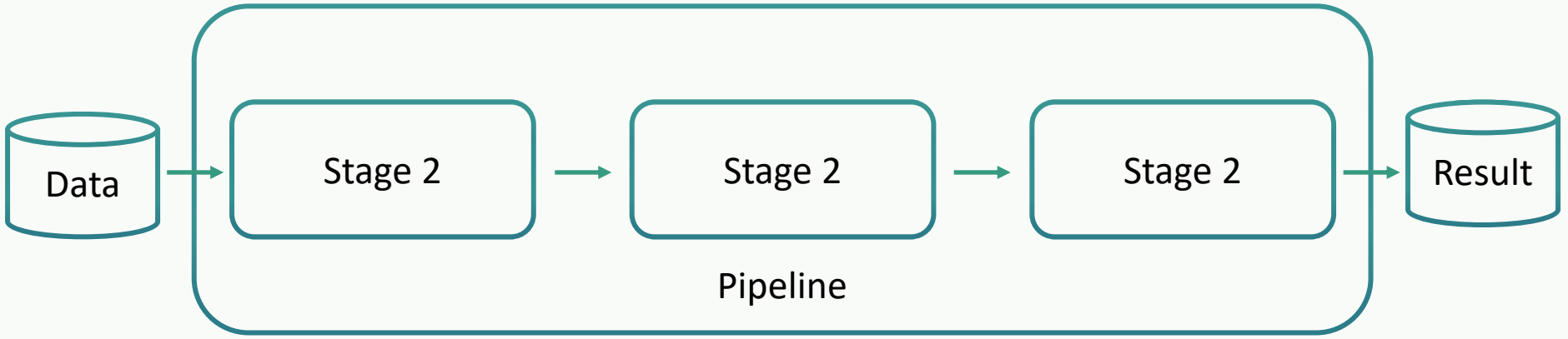
Spark

Фреймворк для распределённой пакетной или потоковой обработки данных

- Распределённые вычисления на кластере
- Реализованные ml алгоритмы
- Инструментарий для Feature Engineering
- API для Java, Scala, Python, R

Spark ML pipeline

Высокоуровневый API для комбинирования различных алгоритмов обработки данных в единый конвейер



- Много готовых стейджей с ml моделями и другой обработки данных
- Можно писать кастомные стейджи
- Возможность сохранения и загрузки конвейеров

Spark ML pipeline. Пример

```
// Создаём 3 этапа обработки данных
val tokenizer = new Tokenizer()
  .setInputCol("text")
  .setOutputCol("words")
val hashingTF = new HashingTF()
  .setNumFeatures(1000)
  .setInputCol(tokenizer.getOutputCol)
  .setOutputCol("features")
val lr = new LogisticRegression()
  .setMaxIter(10)
  .setRegParam(0.001)

// Создаём пайплайн, который состоит из 3 этапов
val pipeline = new Pipeline()
  .setStages(Array(tokenizer, hashingTF, lr))
```

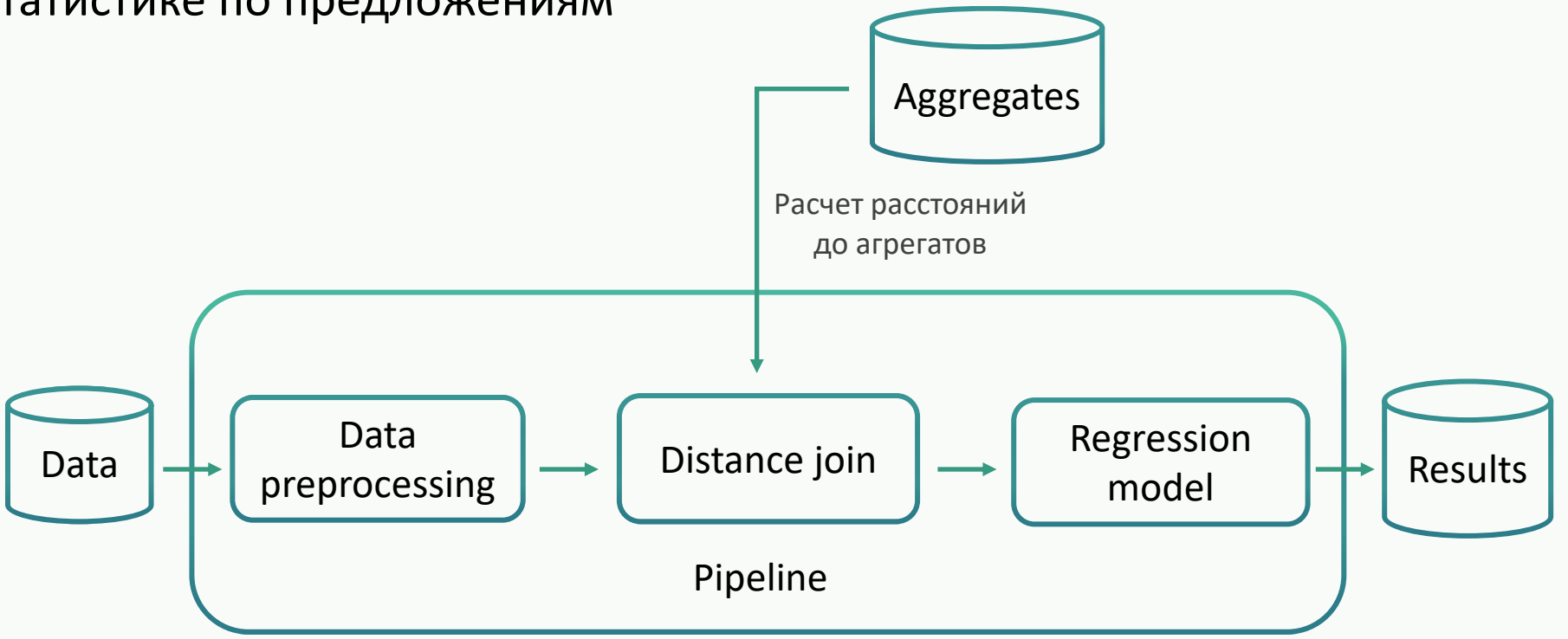
```
// Обучаем наш пайплайн
val model = pipeline.fit(data)

// Сохраняем его на диск
model.write.overwrite().save("/tmp/spark-logistic-regression-model")

// Загружаем
val loadedModel = PipelineModel.load("/tmp/spark-logistic-regression-model")

// Запускаем обработку данных
val result = loadedModel.transform(data)
```

Источник: https://spark.apache.org/docs/latest/ml-pipeline.html

Построение модели

Модель обучается на:

- Различных агрегированных данных о пользователях
- Истории взаимодействия пользователей с предложениями
- Статистике по предложениям

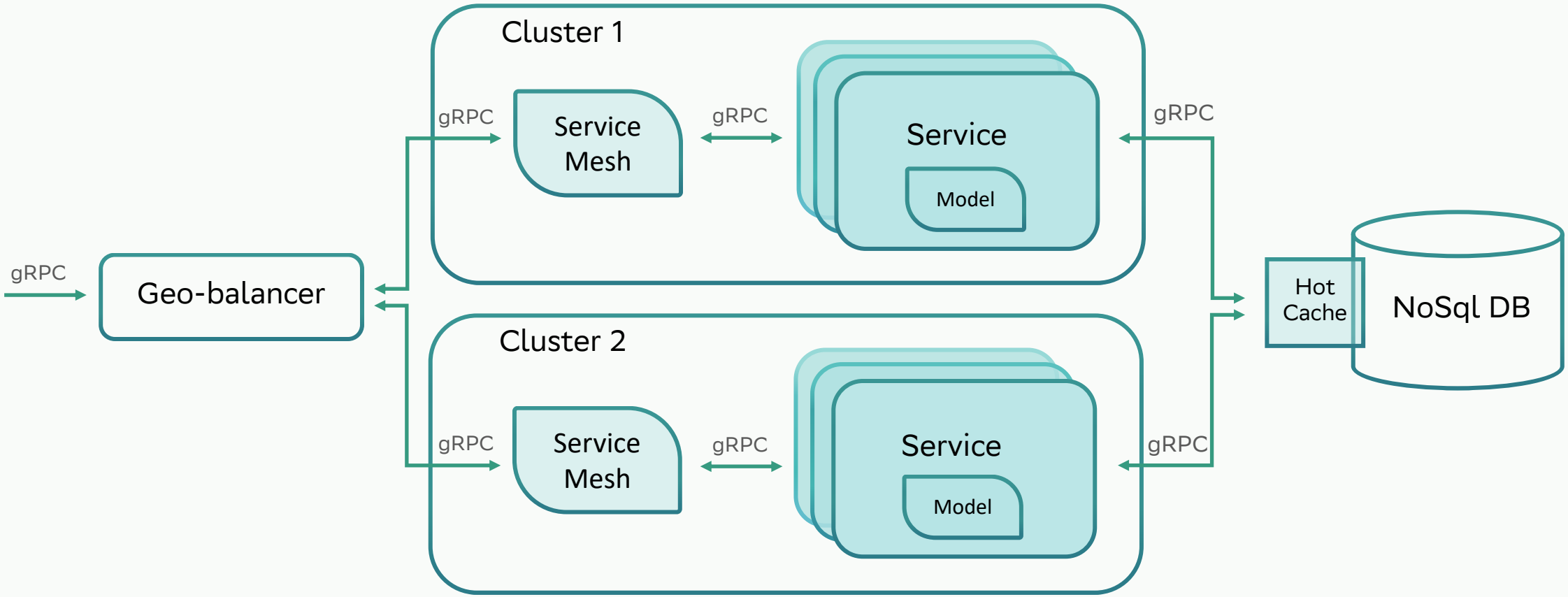


Сервис

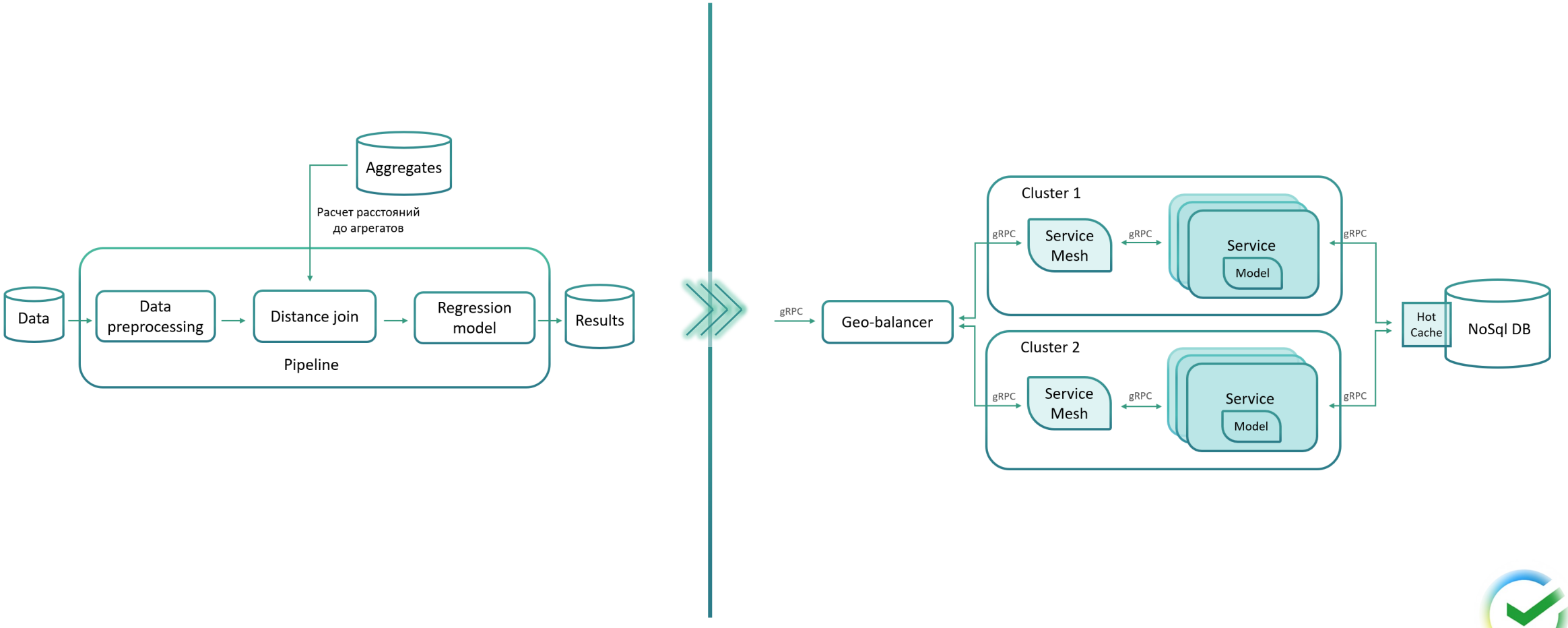
- Обработка в реальном времени запросов на ранжирование
- High-load – 10000 tps
- Low-latency - Время ответа сервиса < 45ms

```
message RecommendationsRequest {
  // Идентификатор пользователя
  UserId userId = 1;

  // Предложения, которые должны отранжировать
  repeated Item items = 2;

  // Другие поля...
}
```

Архитектура сервиса



Переход из мира больших данных в хайлоад



Переписать?

- Долго
- Потенциальные баги
- Требуется экспертиза и в разработке и в ML

Решение 1. SynapseML 

Библиотека от Microsoft для упрощения создания распределённых ml пайплайнов.

- Open-source
- На основе Spark
- Куча всяких ML алгоритмов (Computer Vision, Deep Learning, Text Analitics и т.д.)
- **Деплой моделей в виде сервиса**

Решение 1. SynapseML. Spark Serving

```
val df = spark.readStream
  .server
  .address("localhost", 8888, "my_api")
  .load()
  .parseRequest("my_api", StructType(Array(StructField("foo", StringType))))

// Тут может быть ваша модель
val replies = df.withColumn("fooLength", length(col("foo")))
  .makeReply("fooLength")

replies.writeStream
  .server
  .replyTo("my_api")
  .queryName("my_query")
  .start()
```

Источник: https://microsoft.github.io/SynapseML/docs/Deploy%20Models/Overview/

Пример

```
curl --header "Content-Type: application/json" \
    --request POST \
    --data '{"foo": "abc"}' \
    localhost:8888/my_api


// Response
{"fooLength":3}
```

Решение 1. SynapseML. Spark Serving (continuous)

```
val df = spark.readStream
  .continuousServer
  .address("localhost", 8888, "my_api")
  .load()
  .parseRequest("my_api", StructType(Array(StructField("foo", StringType))))

val replies = df.withColumn("fooLength", length(col("foo")))
  .makeReply("fooLength")

replies.writeStream
  .continuousServer
  .trigger(Trigger.Continuous("1 minute")) // Про триггер будет чуть позже
  .replyTo("my_api")
  .queryName("my_query")
  .start()
```

Источник: https://microsoft.github.io/SynapseML/docs/Deploy%20Models/Overview/

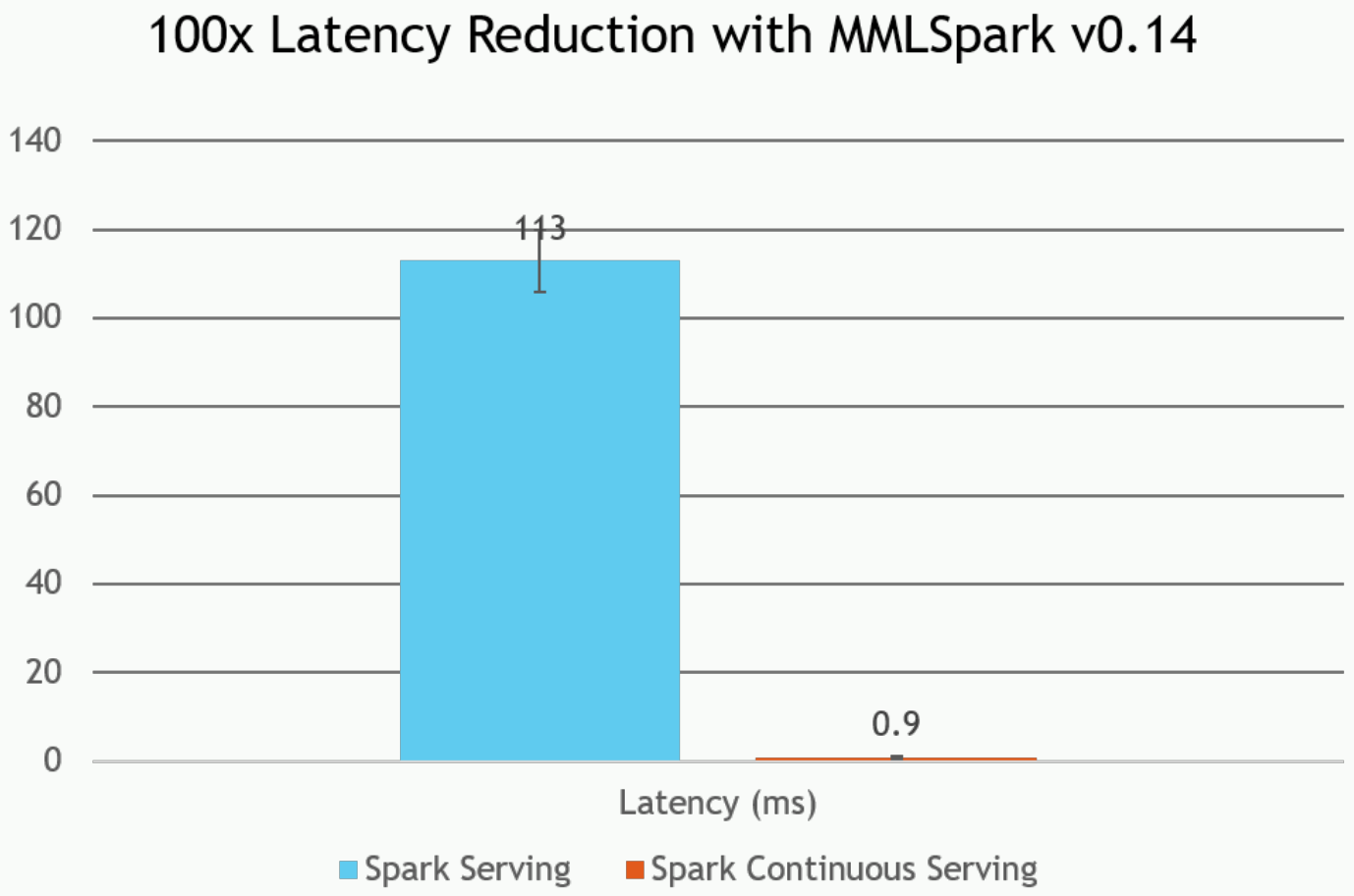
Ограничения Continuous Streaming

- Only map-like Dataset/DataFrame operations are supported in continuous mode, that is, only projections (select, map, flatMap, mapPartitions, etc.) and selections (where, filter, etc.).
- All SQL functions are supported except aggregation functions.

```
// Не работает
df1.join(df2, df1("foo") === df2("text"), "inner")
```

Документация: https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#supported-queries

Решение 1. SynapseML. Spark Serving (continuous)



Источник: https://microsoft.github.io/SynapseML/docs/Deploy%20Models/Overview/

Решение 1. Результаты

- Micro-batch streaming – около 300ms
- Continuous streaming – 10-20 ms
- Нет батч режима

Как это работает

```
val df = spark.readStream
  .continuousServer
  .address("localhost", 8888, "my_api")
  ...
```

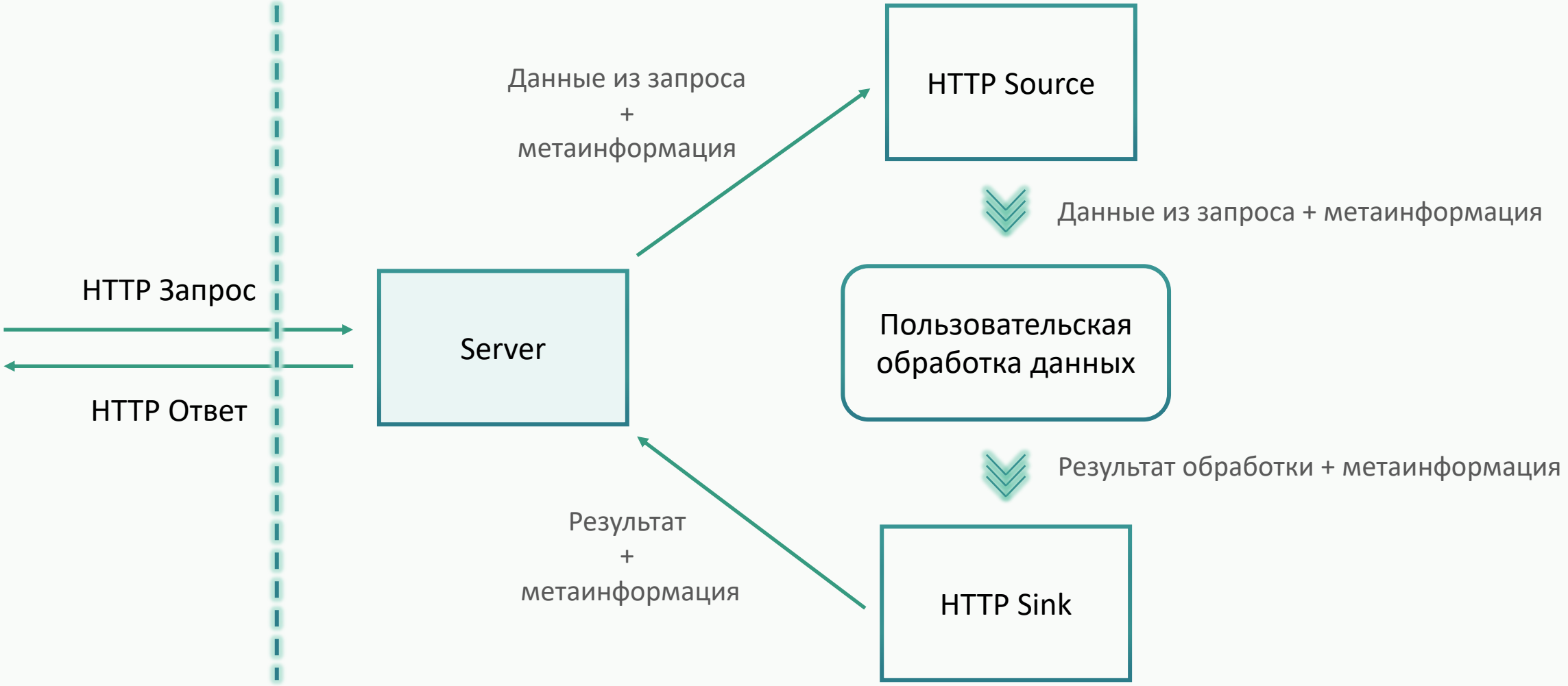
HTTP Source [1]

```
replies.writeStream
  .continuousServer
  ...
```

HTTP Sink [2]

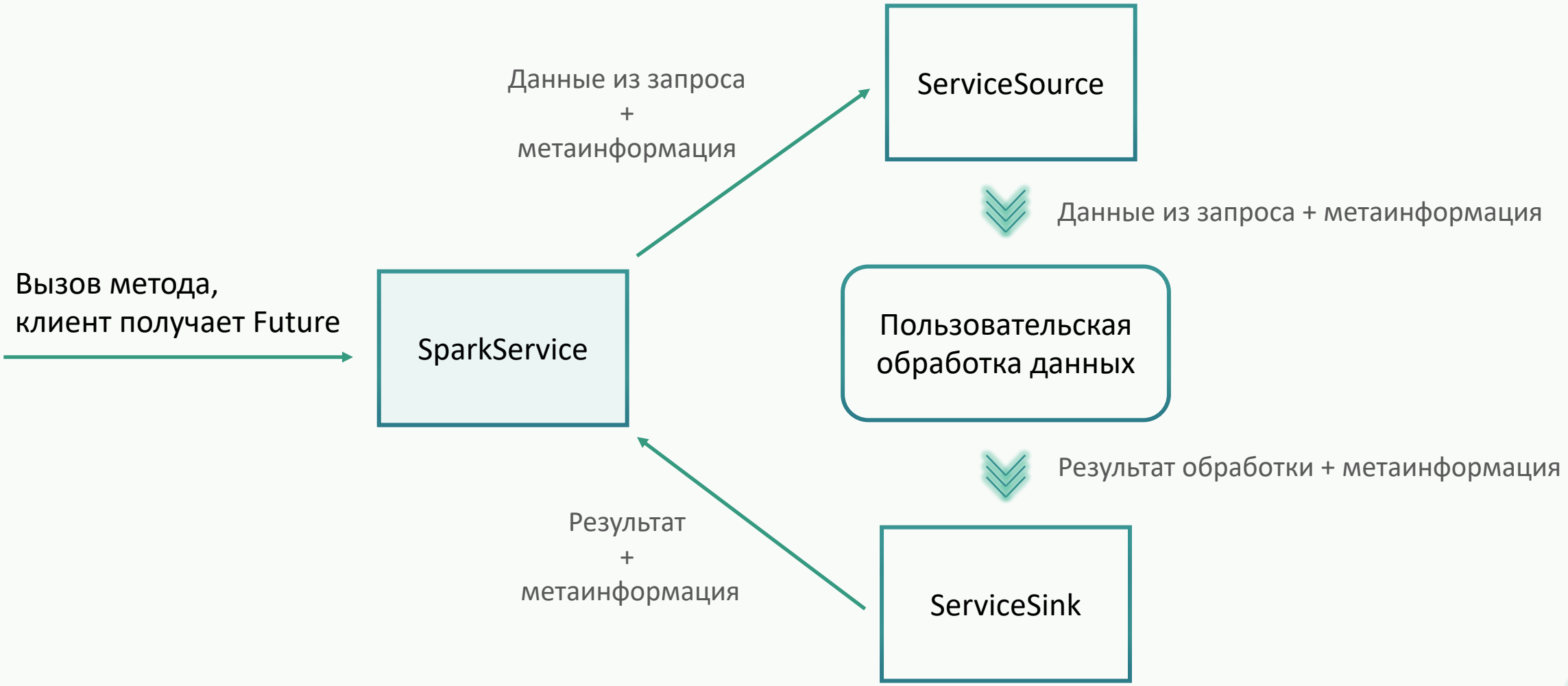
[1] HTTPSourceV2 - https://github.com/microsoft/SynapseML/blob/master/core/src/main/scala/org/apache/spark/sql/execution/streaming/continuous/HTTPSourceV2.scala
[2] HTTPSinkV2 - https://github.com/microsoft/SynapseML/blob/master/core/src/main/scala/org/apache/spark/sql/execution/streaming/continuous/HTTPSinkV2.scala

Как это работает



Сериализация - Сеть - Десериализация

Сделаем аналогично, но без HTTP



Сделаем аналогично, но без HTTP. Source

Как подцепить наш кастомный сурс к спарку:

```
val stream = spark.readStream
  .format(classOf[ServiceSourceProvider].getName)
  .load()
```

Сделаем аналогично, но без HTTP. Sink

Как подцепить наш кастомный синк к спарку:

```
stream.writeStream
  .foreach(new ServiceSink)
```

Сделаем аналогично, но без HTTP. SparkService

- Связующее звено между Source и Sink
- Предоставляет API для запросов клиенту

Тонкие моменты

- Ограничение очереди запросов
- Разогрев
- Таймауты

Trigger.Continuous

```
res.writeStream
  .foreach(new ServiceSink)
  .trigger(Trigger.Continuous("1 hour"))
```

Результаты

Время обработки одного вызова (среднее):

- Без модели - 0.2ms
- С моделью – 14ms

Ссылка на код:

https://github.com/IlyaCES/joker-spark-service

Спасибо за внимание!