Apache Spark: что, зачем и почему?

Сергей Свитков

отдел разработки ПО номер 3

НТЦ ”Протей”

12 июня 2018 г.

Введение

План доклада:

- Существующие проблемы с импортом и обработкой данных с помощью dataproc
- Обзор Apache Spark
- Рассуждения о том, как решить проблемы из пункта 1

DataProc и его минусы

Итак, казалось бы, всё замечательно:

- Зачем тратить время на изучение существующих решений, если можно потратить больше времени на написание своего?
- Можно сделать самый кастомный в мире механизм конфигруации
- Да и вообще, спулеры, фетчеры и всё-всё-всё...
- Ну и, конечно, обработка данных на уровне базы данных! (средствами процедур баз данных, очевидно)

DataProc и его минусы

Однако:

- Один человек не может сделать продукт, по качеству и актуальности не уступающий тому, который делают 50 человек
- Если подобная разработка находится строго в какой-то внутренней среде, она обречена на устаревание
- Изучить существующую технологию проще, чем разобраться с тем, что есть сейчас
- Гораздо больше документации и ответов на часто возникающие вопросы
- При смене кого-то из кадров/срабатывании bus factor/... — найти человека сразу же с нужными навыками невозможно

DataProc и его минусы: итоги

Что имеем:

- Огромную кодовую базу
- Малое количество документации
- Абсолютно приватные знания о проекте и о том, как им пользоваться
- Сложный механизм настройки
- Несмотря на предыдущий пункт, отсутствие гибкости инструмента
- Откровенную сложность работы

Что же делать?

Варианты:

- Провести координальный рефакторинг существующей кодовой базы, ядро выложить в open source
- Реализовать аналогичный инструмент целиком с нуля
- Реализовать новое решение с использованием Apache Spark

Решение с использованием Apache Spark

Это позволит:

- Унифицировать процесс импорта данных
- Вынести обработку данных в код
- Делать всё это многопоточно
- В случае необходимости подключить к проекту нового человека — легко обучить его
- Использовать общеизвестную технологию
- Как следствие — гораздо быстрее находить ответы на свои вопросы

Решение с использованием Apache Spark

Это позволит:

- Унифицировать процесс импорта данных
- Вынести обработку данных в код
- Делать всё это многопоточно
- В случае необходимости подключить к проекту нового человека — легко обучить его
- Использовать общеизвестную технологию
- Как следствие — гораздо быстрее находить ответы на свои вопросы

Apache Spark

- Фреймворк для распределенной обработки больших данных
- Создан в 2012 году, первый официальный релиз в 2014
- Open source, написан на Scala

Apache Spark

Преимущества

- Failure tolerance
- Абстракции данных строго типизированы
- Абстракции данных очень похожи на стандартные коллекции в Scala/Java
- Три типа абстракций: RDD, DataSet, DataFrame, подробнее о них позже
- Топология узлов master/slave
- Кэширование результатов операций
- Catalyst optimizer

Apache Spark

Абстракции представления данных

- RDD:
  - Строго типизированная коллекция
  - Важен порядок операций
  - Никак не оптимизируется автоматически
  - Хорошо подходят для плохо структурированных данных
  - Функциональное API
  - Самый медленный из трёх вариантов

Apache Spark

Абстракции представления данных

- DataSet:
  - Типизированная коллекция
  - Функциональное API
  - Частично оптимизируется автоматически
  - Хорошо подходит для структурированных данных
  - Хороший перформанс, но медленнее, чем DataFrame

Apache Spark

Абстракции представления данных

- DataFrame:
  - Нетипизированная коллекция
  - Взаимодействие осуществляется средствами Spark SQL
  - Полностью оптимизируется с помощью Catalyst
  - Подходит только для структурированных данных
  - Самый быстрый перформанс

Apache Spark

Операции над данными

- Transformations
  - lazy
  - map, flatMap, filter, groupBy, ...
  - Возвращают коллекцию
- Actions
  - eager
  - reduce, fold, agregate, ...
  - Возвращают значение (что-то, но не коллекцию)
- Чтобы применить цепочку трансформаций, нужно вызвать Action

Apache Spark

Почему не Hadoop?

- Hadoop шаффлит данные после каждой операции
- Persistense, т.е. запись промежуточных результатов на диск
- Из-за этого — низкая скорость работы

Apache Spark

Как что-то посчитать?

- Создать SparkJob
- Есть одна master нода и много worker нод
- Master нода содержит SparkContext
- Worker ноды содержат SparkExecutor
- Master нода создает коллекции с данными и распределяет их между worker нодами
- Они коммуницируют через cluster manager
- Между нодами минимизируется передача данных за счёт передачи исполяемого кода

Итоги

Возможно, слишком амбициозное предложение, но всё же:

- Использовать Apache Spark
- С его помощью унифицировать процессы чтения и записи данных
- Для каждого заказчика писать парсер его данных
- Запускать обработку данных не на серверах заказчика, а выделить несколько наших серверов и работать на них
- Это позволит сильно ускорить процесс запуска импорта данных для нового заказчика

Материалы к докладу

- https://www.coursera.org/specializations/scala
- https://stepik.org/course/75/syllabus
- https://stepik.org/course/693/syllabus
- https://github.com/likeanowl/Paper/blob/master/Notes/Sp