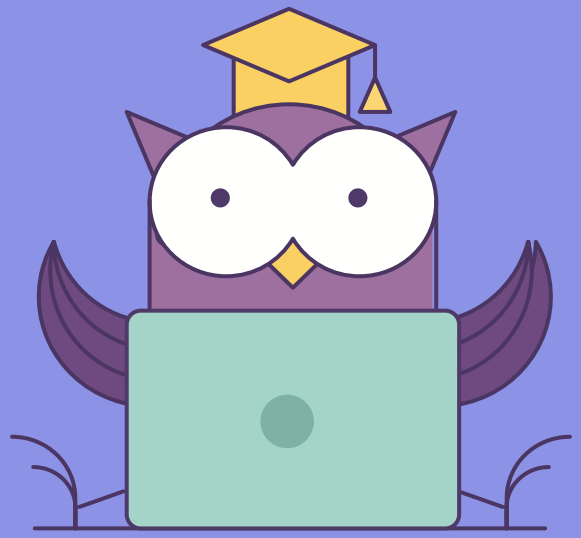
ОНЛАЙН-ОБРАЗОВАНИЕ

Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

OTUS

Apache Spark

Внутреннее устройство

Егор Матешук

Цели сегодняшнего занятия

OTUS

После этого занятия вы будете знать

- Как происходит запуск приложений Spark
- Какая архитектура у приложений Spark
- Какие есть возможности оптимизации и решения типовых проблем в Spark

01

Структура приложения Apache Spark

Немного определений

Давайте еще раз вспомним основные элементы

- Application - приложение, состоит из Driver и Executors
- Driver - приложение, запускающее главный класс, и создающее SparkContext/SparkSession
- Executors - приложения-исполнители, запускающие задачи и хранящие на себе данные
- Application jar - файл, содержащий в себе всё приложение

Немного определений

Давайте еще раз вспомним основные элементы

- Cluster manager - менеджер ресурсов, выделяющий контейнеры для driver и executors (YARN, Mesos, ...)
- Job - задача исполнения графа вычислений
- Stage - этапы вычислений, на которые разбивается job
- Task - конкретный кусочек работы, который будет выполняться на executor (stage состоит из нескольких task)

Запуск на YARN

OTUS

Для запуска приложения на кластере используется spark-submit

```
export HADOOP_CONF_DIR=XXX

./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn \
  --deploy-mode cluster \
  --executor-memory 20G \
  --num-executors 50 \
  /path/to/examples.jar 1000
```

Запуск на YARN

- --driver-cores / --executor-cores - количество ядер для каждого из элементов приложения (на контейнер)
- --driver-memory / --executor-memory - количество памяти для каждого из элементов приложения (на контейнер)
- --queue - очередь в YARN, в которой будет выполняться приложение
- --num-executors - количество executors (может быть динамическим)

Uber JAR

Spark-приложение упаковывается в uber-jar, содержащий необходимые зависимости.

Его можно располагать как на локальной FS, так и на HDFS.

Такой jar можно собрать командой

```
sbt assembly
```

(для этого нужен плагин sbt-assembly)

Дополнительные зависимости

Если вы не хотите тащить с собой лишние зависимости, есть три возможности:

- --jars - указание пути к дополнительным джарникам
- --packages - подключение зависимостей из удаленных репозиториев (Maven)

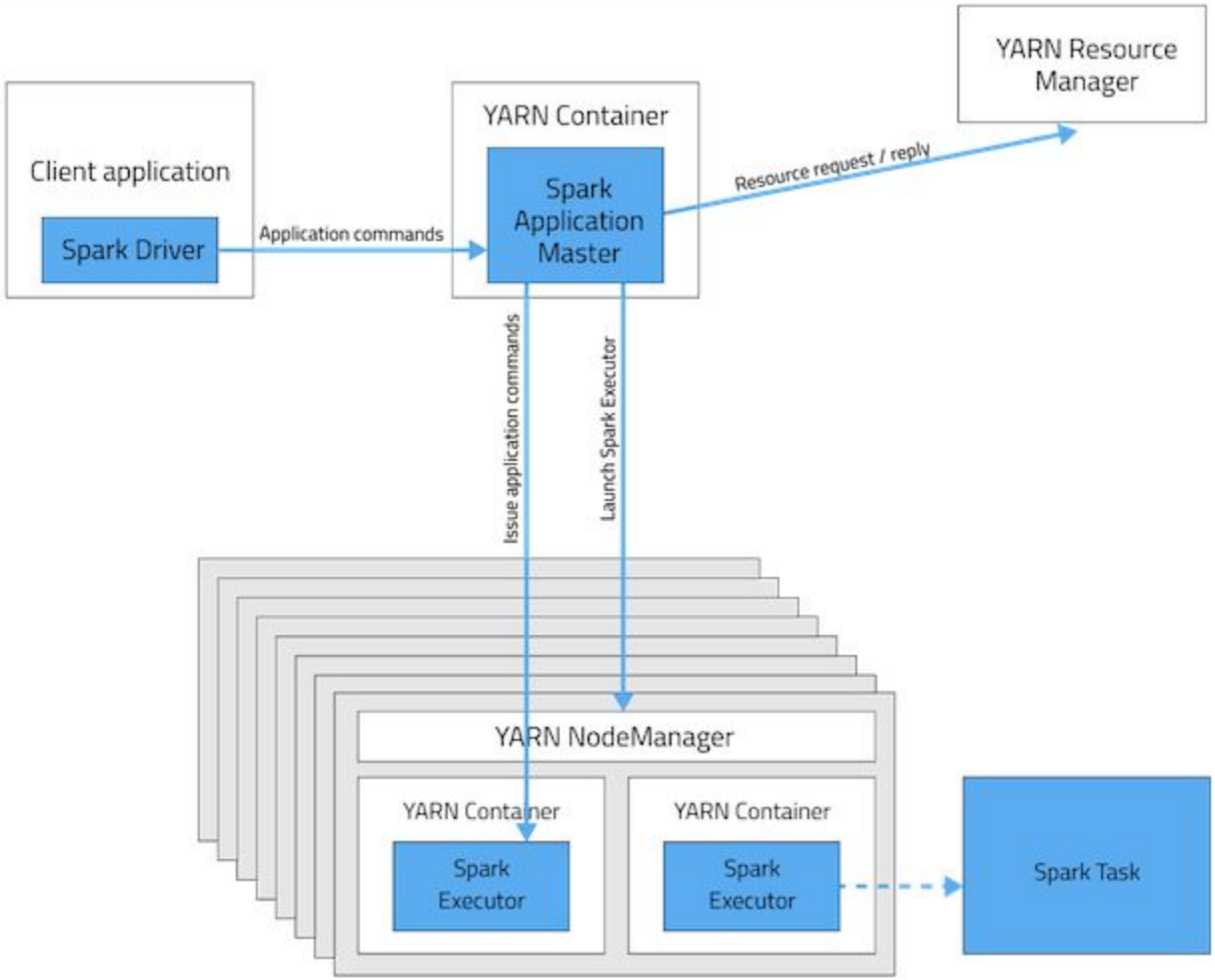
  ```
  --packages datastax:spark-cassandra-connector_2.11:2.0.7
  ```
- CLASSPATH - переменная окружения, в которой можно указать дополнительные джарники

Deploy mode

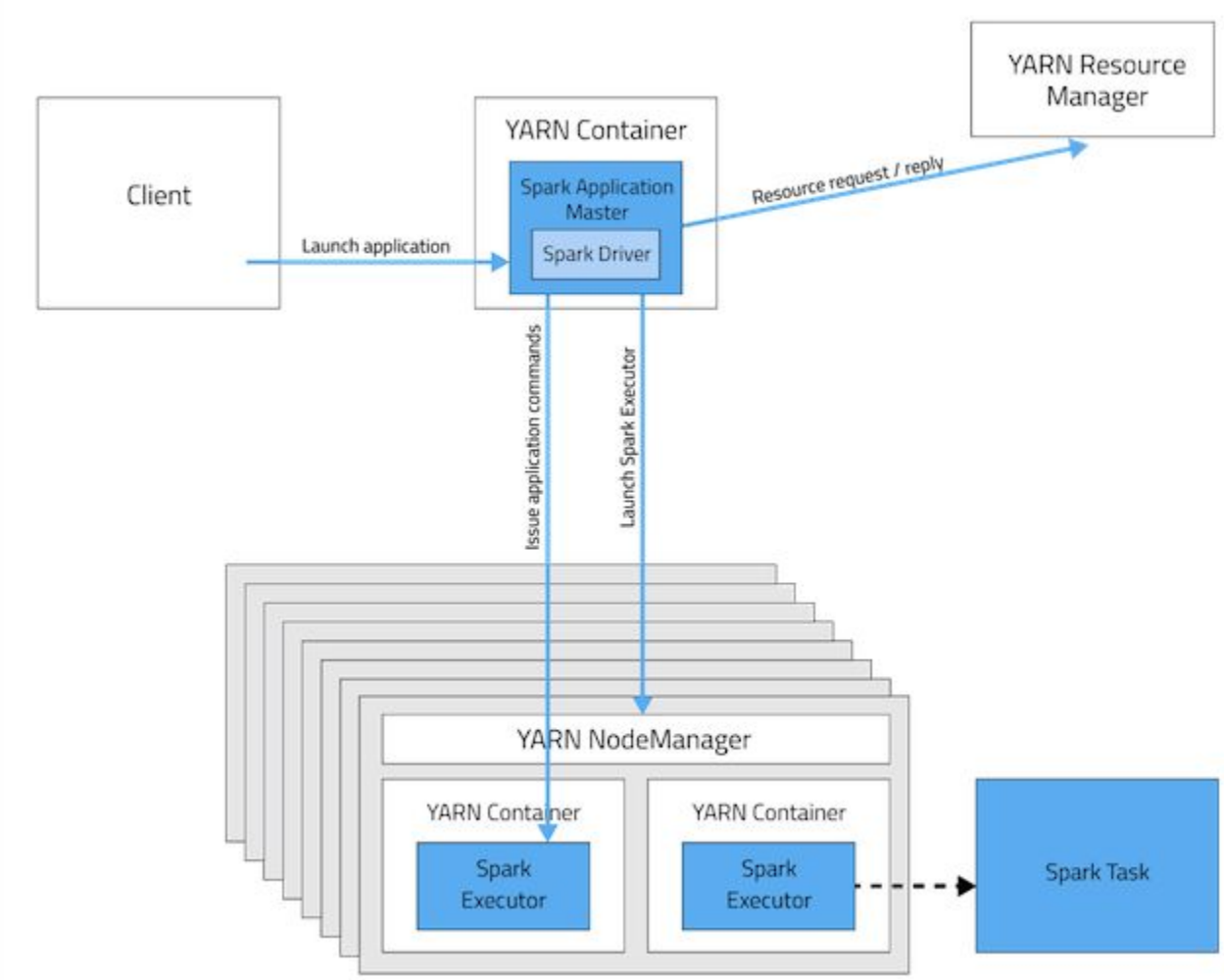
Есть два режима деплоя приложения:

- client - драйвер запускается локально, executors - на кластере
- cluster - драйвер, как и executors, запускается на кластере

Deploy mode - Client



Deploy mode - Cluster



Task locality

При выполнении задач, они выбираются каждым экзекьютором в следующем порядке приоритета

- PROCESS_LOCAL
- NODE_LOCAL
- RACK_LOCAL
- ANY

02

Оптимизация Spark

Часто встречающиеся проблемы

- Нехватка памяти
- Мелкие файлы
- Перекос в партициях данных
- Медленные джойны
- Broadcast больших кусков
- Слишком частое использование UDF
- Переиспользование общих ресурсов (соединений, парсеров, ...)

Нехватка памяти

Симптом: Container killed by YARN for exceeding memory limits

Как лечить:

- Перестраивать граф
- Увеличивать объем памяти
- “Склеивать” экзекьюторы
- Тюнить сериализацию и лимиты памяти

Нехватка памяти

Перестраивать граф:

Чтобы увидеть граф - идем в UI или .explain

Потребление памяти часто растет на shuffle.

Соответственно, нужно максимально сдвинуть этапы сортировок, группировок и других перемещений данных между нодами в конец вычислений.

Нехватка памяти

Увеличивать объем памяти:

- Увеличиваем количество экзекьюторов
- Увеличиваем объем памяти на экзекьютор с помощью --executor-memory

Нехватка памяти

“Склеивать” экзекьюторы:

Поскольку память задач ограничивается памятью EX, то можно вместо двух EX 2x4G использовать один EX 4x8G. И таким образом сгладить использование памяти отдельными тасками.

Этот способ подходит, если проблему вызывают разные по размеру партиции.

Нехватка памяти

OTUS

Тюнить сериализацию и лимиты памяти:

Можно подкрутить некоторые настройки спарка для оптимизации памяти

- Включение KryoSerializer

```
conf.set("spark.serializer",
"org.apache.spark.serializer.KryoSerializer")
```

- Изменение spark.yarn.executor.memoryOverhead

Мелкие файлы

Симптом: Вы провели вычисления и сохранили результат, но в папке результата оказалась куча мелких файлов.

Как лечить:

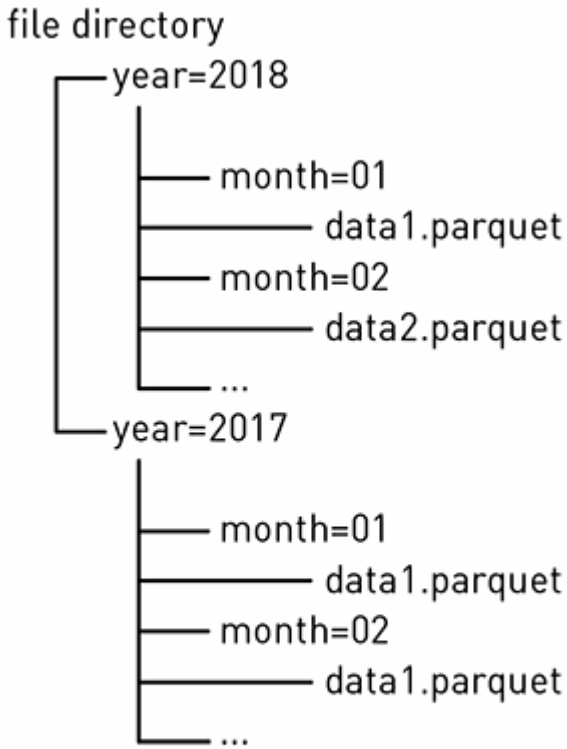
```
df.repartition(1)
  .write.parquet("hdfs:///parquet-files/")
```

или

```
df.write.partitionBy("date")
  .parquet("hdfs:///parquet-files/")
```

Отступление: Партицирование папками

```
df.write.partitionBy("year", "month")
  .parquet("hdfs:///parquet-files/")
```



Перекос в партициях данных

OTUS

Симптом: Некоторые таски выполняются намного дольше других.

Как лечить:

- Оптимизировать данные на этапе записи (выровнять партиции или разбить на большее число партиций)
- Во время вычислений использовать repartition

Медленные джойны

Симптом: Очень долго выполняются операции join

Как лечить:

- Если один из датасетов намного меньше другого, то можно использовать broadcast

Broadcast больших кусков

Симптом: Джоба “зависла” - активных тасок нет, но вычисления дальше не идут

Как лечить:

- Убрать бродкаст с больших датасетов

Слишком частое использование UDF

OTUS

Симптом: Джоба работает медленно. Характерных признаков нет, поэтому это скорее рекомендация.

Кастомные функции требуют постоянных сериализаций-десериализаций, которые обходятся дорого.

Если есть возможность, лучше свести всё к стандартным функциям Spark из

`org.apache.spark.sql.functions`

Переиспользование общих ресурсов

OTUS

Симптом: Тормоза на шаге map

Как лечить: Использование .mapPartitions

Переиспользование общих ресурсов

OTUS

```
.map { x =>
  val conn = new Connection()
  // использование conn для обогащения x
  conn.close()
}

.mapPartitions { xs =>
  val conn = new Connection()
  for (x <- xs) {
    // использование conn для обогащения всех xs
  }
  conn.close()
}
```

Проблемы сериализации

Симптом: Task not serializable:

java.io.NotSerializableException

Как лечить:

В stack trace будет указан элемент, который не удалось сериализовать. Этот элемент нужно создавать на экзекьюторах, а не на драйвере. Например, внутри mapPartitions

Accumulators

Accumulators - особый вид переменных, которые позволяют считать общие метрики по всему кластеру

```
val accum = sc.longAccumulator("My Accumulator")

sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum.add(x))

println(accum.value)
```

03

Подведем итоги

Цели сегодняшнего занятия

OTUS

После этого занятия вы будете знать

- Как происходит запуск приложений Spark
- Какая архитектура у приложений Spark
- Какие есть возможности оптимизации и решения типовых проблем в Spark

Заполните, пожалуйста, опрос в личном кабинете!

Полезные ссылки

- Документация Spark
- Примеры по Spark
- Отличный блог по Scala с кучей рецептов
- Книга по Scala (Essential Scala Book)
- Optimizing Apache Spark SQL Joins

Егор Матешук

egor@mateshuk.com

Спасибо
за внимание!