Spark

Количество worker’ов(потоков) = (P = executors\*cores)

1. Параметры запуска Spark-submit

Флаг --master служит для определения URL кластера для подключения. В данном примере URL spark:// означает, что кластер действует под управлением Spark Standalone. Помимо этого, сценарию spark-submit можно передать множество различных параметров, которые будут описывать некоторые особенности запускаемого приложения. Эти параметры можно разделить на 2 категории:

* информация для планирования (например, объем ресурсов, необходимых для выполнения задач);
* сведения о зависимостях приложения (библиотеки или файлы для подгрузки).

Флаг --master имеет следующие возможные значения:

* spark://host: port – адрес кластера c диспетчером Spark Standalone и порт, по которому он подключен (по умолчанию 7077);
* mesos://host:port – адрес кластера c диспетчером Mesos и порт, по которому он подключен (по умолчанию 5050);
* yarn – адрес кластера с диспетчером [YARN](https://spark-school.ru/wiki/yarn/). Перед запуском требуется обязательно определить переменную окружения [HADOOP](https://spark-school.ru/wiki/hadoop-wiki/)\_CONF\_DIR, в которой необходимо указать путь к каталогу с настройками [Hadoop](https://spark-school.ru/wiki/hadoop-wiki/" \t "_blank);
* local – запуск на локальной машине;
* local(N) – запуск на локальной машине с N ядрами;
* local(\*) – запуск на локальной машине с использованием всех ядер.

Кроме флага --master сценарий spark-submit имеет еще несколько часто используемых флагов, среди которых следующие:

* --deploy-mode – определяет, как должна запускаться программа: локально (client) или на каком-либо из узлов кластера (cluster). По умолчанию используется режим client;
* --class – определяет главный класс для приложения (в языках JAVA или Scala);
* --name – имя приложения, отображаемое в веб-интерфейсе Spark;
* --files – файлы для размещения в рабочем каталоге программы;
* --py-files – файлы для добавления в переменную окружения PYTONPATH;
* -- executor-memory – объем памяти в байтах, которая выделяется для исполнителЯ;
* --driver-memory – объем памяти, которая выделяется для driver-программы.

2. 4 Метода оптимизации

2.1 repartition

разорвать этап на два, используя функцию repartition(N), которая приводит к перетасовке в случайном порядке, создавая на выходе N блоков данных, примерно равных по размеру. А так как она производит перетасовку (Shuffle), значит, начинается новый этап.

coalesce(N). Этот метод просто зачисляет каждый блок нашего набора в один из N новых блоков. То есть реально coalesce(), в отличие от repartition(), не приводит к перетасовке, а следовательно не разрывает этап, просто делает так, что на нашем этапе будет N блоков данных

2.2 cache()persist()

* **cache()** – cохраняет DataFrame с уровнем хранения по умолчанию (MEMORY\_AND\_DISK) в соответствии со Scala 0 **[2]**;
* **persist()** – устанавливает уровень хранения для содержимого DataFrame между операциями после первого вычисления.
* DISK\_ONLY: данные сохраняются на диске в сериализованном формате
* MEMORY\_ONLY: данные сохраняются в оперативной памяти в десериализованном формате
* MEMORY\_AND\_DISK: данные сохраняются в оперативной памяти, и если памяти недостаточно, вытесненные блоки будут сохранены на диске
* OFF\_HEAP: сегмент памяти за пределами JVM, который иногда используется виртуальной машиной Java

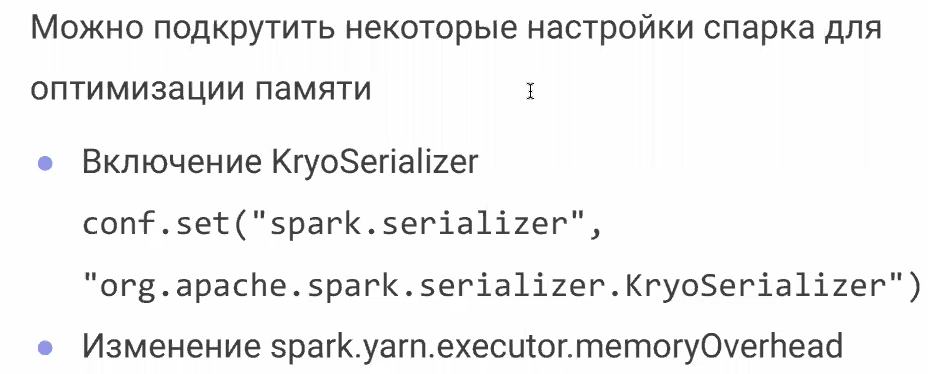


cache и leteral view не позволяют оптимизатору дальнейшие обработки спускать ниже уровня cache’а.

2.3 BROADCAST

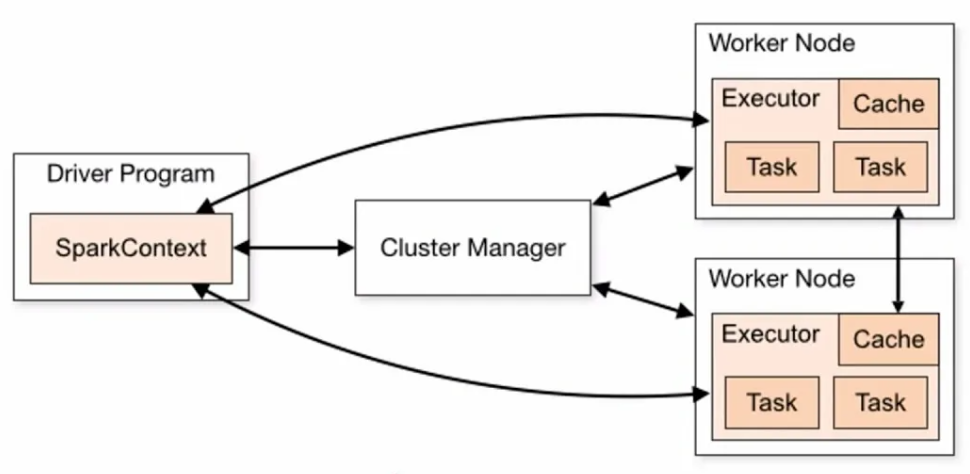
В случае Hive и Spark прелесть этого метода состоит в том, что мы можем переслать на каждый из узлов не только таблицу, но вообще говоря любую структуру данных.

2.4 Тюнить сериализацию и лимиты памяти



3. Обычными причинами, приводящими к OutOfMemory OOM (недостаточно памяти) драйвера, являются:

* rdd.collect()
* sparkContext.broadcast
* Низкий уровень памяти драйвера, настроенный в соответствии с требованиями приложения
* Неправильная настройка Spark.sql.autoBroadcastJoinThreshold



**3. SPILL**

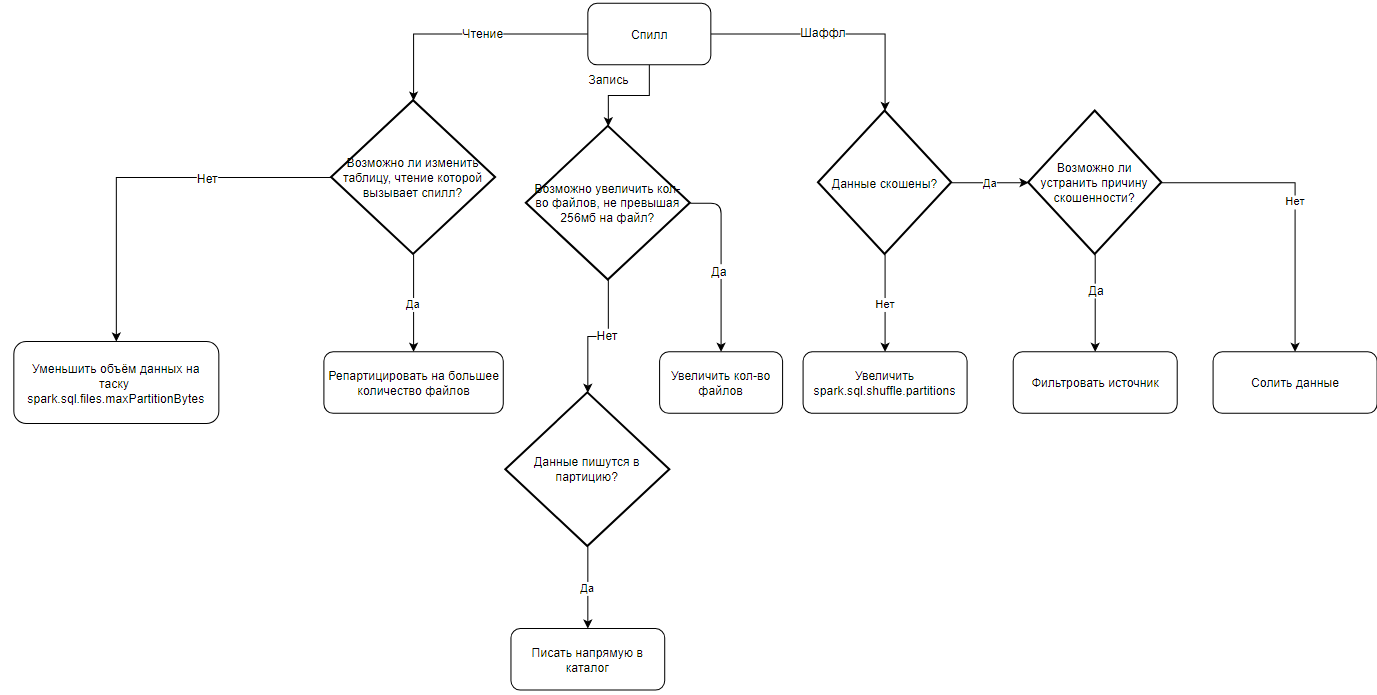
Каждой задаче назначается раздел памяти для обработки данных. Если данных слишком много, и они не помещаются в выделенный для задачи раздел памяти, чтобы не потерять вычисления, данные записываются (переливаются) на диск. spill можно рассматривать как встроенный в Apache Spark способ устранения ошибки нехватки памяти ([OutOfMemory, OOM)](https://www.bigdataschool.ru/blog/memory-leaks-oom-in-spark-apps.html" \t "_blank), он сам может стать причиной снижения производительности приложения

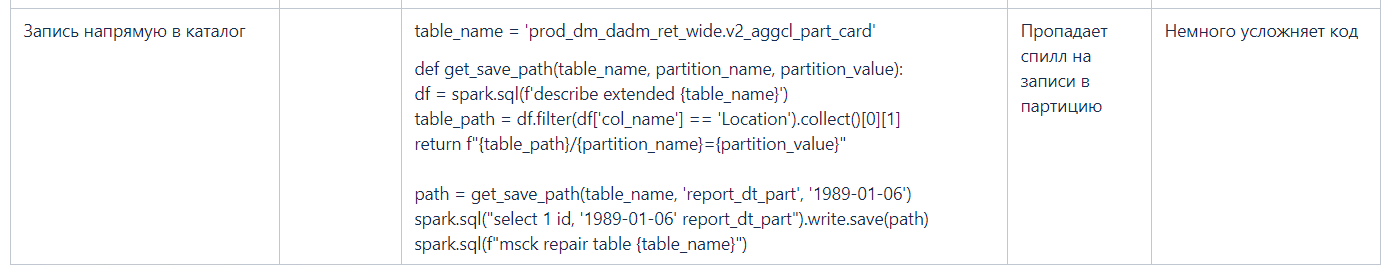
**переброс** или **spill**– это термин для обозначения процесса перемещения RDD из RAM на диск, а затем снова обратно в RAM

Возникает как заканчивается storage memory и executor memory

Есть 3 вида Спиллов:

* + 1. Спилл на чтение
    2. Спилл на запись
    3. Спилл на шаффле

**значение параметра spark.sql.files.maxPartitionBytes** – количество байтов для упаковки в один раздел при чтении файлов формата [Parquet](https://www.bigdataschool.ru/wiki/parquet" \t "_blank), JSON и [ORC](https://www.bigdataschool.ru/wiki/orc). По умолчанию это свойство задано 128 МБ. Если установить раздел, считываемый Spark намного больше, например, 1 ГБ, активное поглощение может не вызвать утечку памяти, но вызовет spill-эффект



как избежать spilов

* уменьшить размер каждого раздела, увеличив их количество;
* задать значение параметра sql.files.maxPartitionBytes;
* явно перераспределить данные с помощью методов repartition() или coalesce, чтобы устранить перекос и сделать их распределение по кластеру более равномерным. Также можно попробовать криптографический метод модификации хэш-функции
* изменить значение параметра spark.sql.shuffle.partitions — количество разделов для использования при перетасовке данных для объединений или агрегатов, по умолчанию равен 200.

**4. Виды операций**

**Сортировки:**

**In-Memory Sorting**

Когда данные помещаются в память, Spark предпочитает timsort по партициям. Timsort - это гибридная стабильная сортировка, основанная на сортировке слиянием и сортировке вставкой.

Шаги in-memory sorting:

1. Data shuffled to destination partitions
2. Inside each partition, timsort runs to locally order the data
3. Partitions concatenated to produce globally sorted order

Timsort обеспечивает быструю производительность O(n log n). Spark также использует оптимизацию, такую как несогласованный доступ к памяти (misaligned memory), для дальнейшего ускорения сортировки.

**Disk-Based Sorting**

Для данных не помещающихся в памяти, Spark сливает данные на диск (spill) используя binsort:

1. В каждой партиции, данные разбиваются на чанки(chunks) которые сортируются используя timsort
2. The chunks are collected into larger bins also stored on disk
3. Bins merged to produce final sorted output

Bin packing helps minimize expensive disk reads and writes. Spark intelligently tunes parameters like the chunk and bin sizes

**JSON**

//mapPartiotions – тот же map только для всей партиции. Например открывать коннект не для каждой записи, а для пачки

Прочитать**:**

spark.read.option("multiLine", "false").json("./dataset/hardStruct.json")

multiline – если true пытается всю запись распознать за одну строчку

Записать**:**

spark.table("pc").write.format("json").save("/dataset/testJS.json")

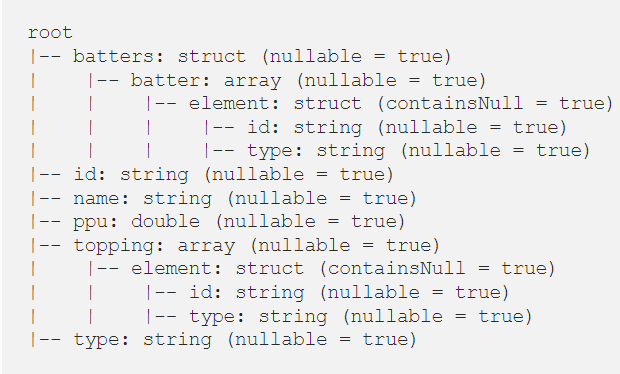
будет не файл «testJS.json», а папка testJS.json с файлами внутри, но для чтения json можно ссылаться на данную папку

json используют для сложных структур данных например:

(это одна строчка)

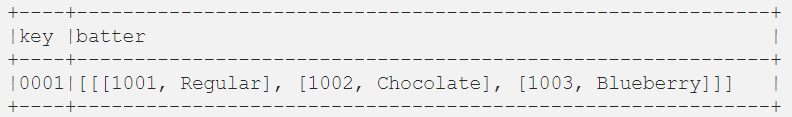


df.printSchema():

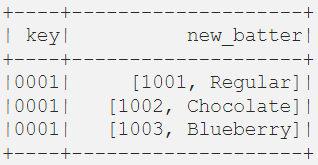


Далее можно расскрывать, превращать в таблицу:

1.



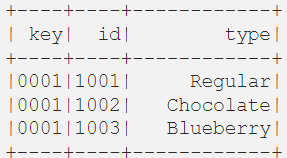
2.



Explode – каждое значения из массива комбинирует со всеми строчками (декартовое произведение)

3.

\*-разбивает массив на столбцы



**SparkStreaming**

<https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c> - рабочий код

другие статьи используют sparkConf, но его нет в новых версиях

