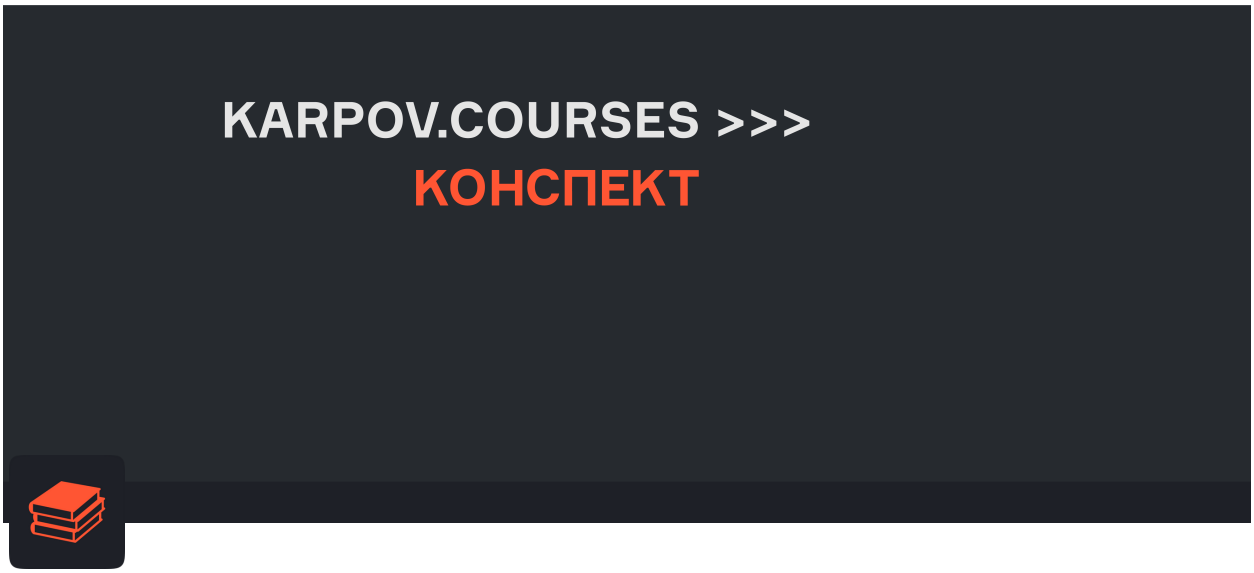


> Конспект > 5 урок > Основы Apache Spark

> Оглавление

> Основные идеи

Spark - эффективный фреймворк для обработки больших данных.

- Эффективная распределенная DAG (directed acyclic graph) модель вычислений (не только MapReduce)
- "Ленивая" модель вычислений. Она позволяет не выполнять операции над объектами, которые дальше нигде не будут использованы. Таким образом лишние вычисления пропускаются
- Гибкие механизмы управлению памятью:
  - предпочтение хранения данных в памяти (это эффективнее с точки зрения доступа и производительности)
  - возможность сброса данных на диск при нехватке памяти (это негативно сказывается на производительности, но зато позволяет не падать джобам, которые не помещаются в оперативную память)
  - возможность кеширования данных в памяти, на диске и комбинировано
  - различные форматы сериализации данных
- Наличие API для Scala, JAVA, Python, R
- Единый API для batch & streaming processing.

> RDD

RDD (Resilient Distributed Dataset) - распределенный набор данных, чьи особенности заключаются в:

- Неизменяемый детерминированный набор данных (fault tolerance). Для изменения, придется выполнить операцию, которая приведет к появлению нового RDD. Такие четкие правила преобразования данных из начального состояния в конечное позволяют при выходи из строя некоторого шага пайплайна откатиться на шаг назад и заново перевыполнить упавшие операции.
- 2 вида операций:
  - transformation - изменение существующего RDD (возвращает другой RDD, например map, filter, join)
  - action - терминальная стадия, приводящая к тому, что накопленные ленивые вычисления начинают свое выполнение (инициация процесса

вычисления, например save, count, collect)

- Партиционированная модель хранения и обработки. Каждая партиция - это единица обработки данных.
- Различные способы кеширования (memory, disk, memory&disk, external*)

> Распределенная модель вычислений в Spark

1. Строится эффективный DAG
2. Запускается распределенная модель вычислений. Используется принцип Master - slave, где master - это некая driver программа, а slave - executor.

Driver программа запускается на ApplicationMaster. Она:

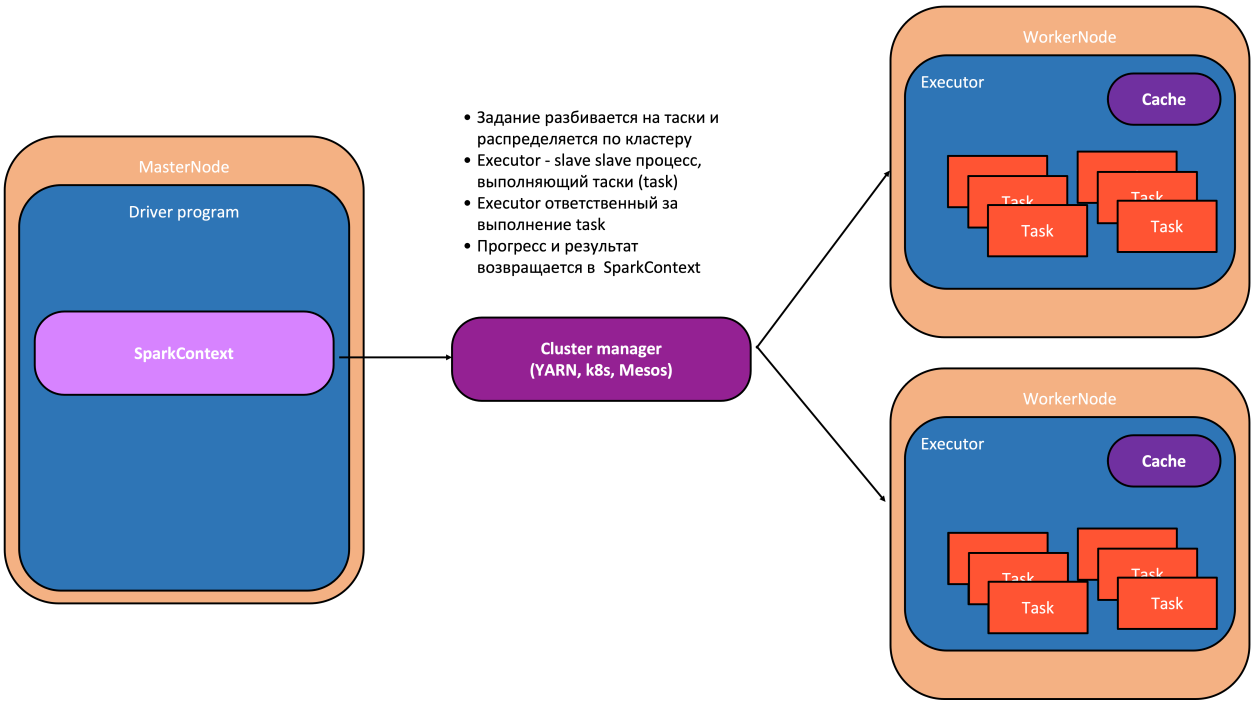
- создает SparkContext/SparkSession - абстракция, позволяющая обеспечить единую точку входа для выполнения рапределенных вычислений. SparkContext позволяет писать писать код так, как будто мы пишем однопоточную программу. Она потом будет преобразована в набор независимых шагов, которые могут быть обработаны параллельно
- распределяет задания между executors
- планирует и отслеживание прогресс выполняемых стадий

Slaves - executors - запущенные java-процессы на различных машинах в кластере, выполняющие параллельную обработку данных.

Архитектура задания

Иерархию выполнения заданий можно описать как:

1. имеем написанное приложение
2. внутри него запускаются задания (job)
3. джобы состоят из стадий (stage)
4. каждая стадия состоит из набора атомарных операция (task)



Как устроена архитектура заданий:

Имеем сервер (MasterNode). На нем запускается Driver, который создает SparkContext.

SparkContext при планировании модели вычислений обращается к Cluster manager (некая абстракция, котороя может быть, например YARN, k8s, Mesos) с целью создания executors.

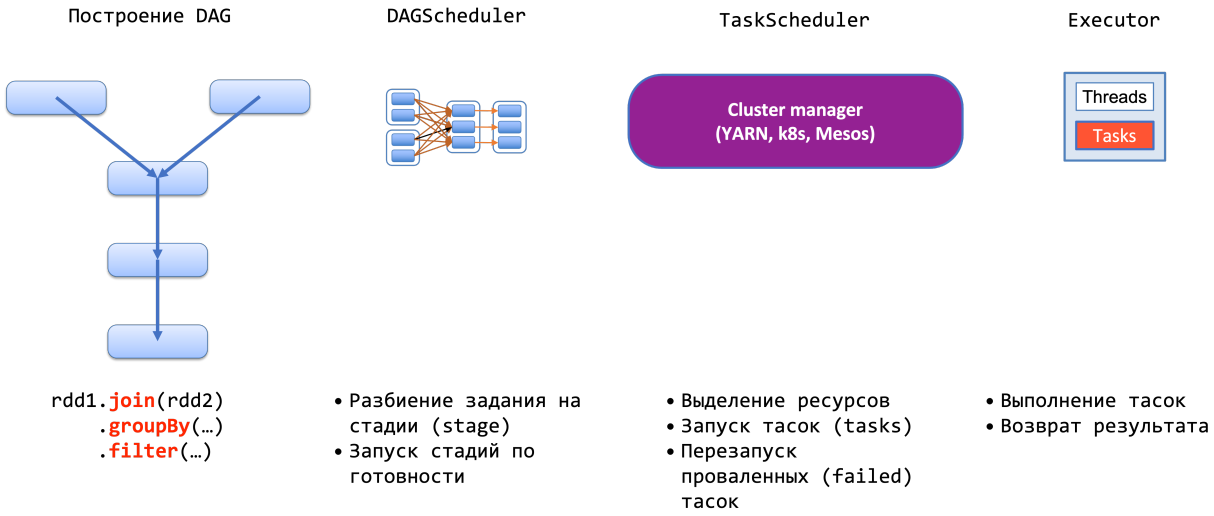
Driver производит планирование вплодь до тасок, на которые разбивается наше задание. Таски далее распределяются внутри кластера между executors.

Executor - slave процесс, выполняющий таски. Если таска уже была выполнена и вычисления есть в памяти, он их сразу заберет. Если данных в памяти нет, то executor должен их получить либо забрать данные с другого executor, либо получить из внешнего хранилища. Driver программа следит за состоянием executors (может из создавать, убивать, перезапускать, переназначать задания итп). Если executor не справляется с выполнением тасок, то driver должен либо перезапустить его на другом сервере и повторить все вычисления, либо же завершить выполнение джобы с ошибкой.

Driver постоянно отслеживает прогресс вычислений всех executors, собирает статистику, счетчики, данные о производительности и различную служебную

информация. Все это возвращается в driver через SparkContext.

> Планировщик



Рассмотрим следующий пример:

Пусть у нас есть некоторый код, который должен выполнить определенный набор операций.

1. Строится DAG.
2. DAG передается в DAGScheduler. Строится конкретный последовательный алгоритм того, как необходимо загружать данные, как их преобразовывать и каким образом будет получен результат. Далее выявленные стадии (stage) по очереди начинаю запускаться. Каждая стадия состоит из конкретного набора тасок. Так же на этом шаге происходит назначение конкретных тасок на executors.
3. Данные из п2 передаются на TaskScheduler - планировщик запуска и мониторинга атомарных процессов обработки данных (тасок). Происходит взаимодействие с Cluster Manager, выделяются вычислительные ресурсы. Внутри контейнеров запускаются таски, происходит обработка информации. В случае ошибки, планировщик пытается перезапустить проваленные таски, заданное количество раз, если уже количество перезапусков превышает

установленное максимально значение, то stage, у которого провалена таска, так же считается проваленным и соответствующая джоба будет аварийно завершена.

4. Таски приходят на executors. Распараллеливание тасок внутри executor-ов тоже возможно, если таски в принципе можно обрабатывать параллельно и на это хватает ресурсов. После выполения executor возвращает результат, который будет дальше подан на вход другим excutor-ам или же сохранит его, или выведет на консоль.

> Экосистема

Spark поддерживает API для Scala, JAVA, Python, R.

Spark может работать:

- на локальной машине в Standalone режиме. Так обычно работает тестирование, потому что это позволяет не поднимать настоящий кластер
- в YARM
- в Mesos
- K8S

Spark состоит из:

- Spark core - ядро, в которое входит все базовые примитивы (RDD, операции, преобразования, модели вычислений)
- Spark SQL - отвечает за то, чтобы было максимально удобно работать со структурированными/полуструктурированными данными, используя классические операции (join, filter, groupby...)
- Spark Streaming - набор библиотек и API, предназначенных для поточной обработки данных
- Spark MLLib - набор библиотек, связанных с машинным обучением, которые позволяют использовать классические алгоритмы МО в распределенной модели вычисления
- GraphX - набор библиотек и API, предназначенных для работы с графовыми данными

Spark core

выполняет низкоуровневое управление процессами вычислений:

- Движок для параллельной распределенной обработки
- Запускает и отслеживает статусы заданий
- Управление памятью
  - сериализация
  - кеширование
- Восстановление после сбоев
- Координация выполнения тасок

Spark SQL

- Движок для работы со структурированными данными
- Поддержка различных API
  - DataFrame API
  - DataFrame DSL (domain-specific language) - более высокоуровневый по сравнению с DataFrame API
  - Spark SQL & HQL
- Уровни API
  - На уровне языка (Scala, Java, Python, R, HiveQL, R)
  - RDD / DataFrame (RDD базовая абстракция, многие классические операции на DataFrame-ах делаются проще и эффективнее)
  - DataSource (JSON, ORC, Hive, Cassandra...). Все внешние источники данных, с которых мы что-то читаем или на которые записываем, для Spark - это DataSource. Используя такую высокоуровневую абстракцию, Spark позволяет переводить API DataSource в методы, специфичные для конкретных источников данных
- Позволяет обрабатывать данные из различных источников в единой модели (то есть используя унифицированные методы, способы обработки)

Spark Streaming

Это часть Spark отвечающая за поточную обработку данных. Системы поточной обработки делятся на те, которые основаны на событийном подходе и на те, которые работают по принципу micro batch (

- Движок для работы с потоковыми данными
- Работы по принципу micro batch
- Позволяет комбинировать потоковые данные со статистическими (например, условно real-time данные с данными из внешних хранилищ или данными, закэшированными в нашем приложении)
- Унифицированный API
- Различный подход к обработке данных
    - Spark Streaming (DStreams) - входящий поток данных разбивается на небольшие куски, далее каждый из кусков обрабатывается последовательно
    - Structures Streaming - входящий поток данных рассматривается как бесконечный dataframe, то есть таблица, в попадают данные и по мере их поступления они сразу обрабатываются. При таком подходе между входом и выходом данных не могу проводиться терминальные операции, информация может только фильтроваться/обогащаться.

Spark MLLib

- Движок machine learning
- Регрессия
- Кластерищация
- Классификация
- Коллаборативная фильтрация
- Создание ML Pipelines
- Специализированные математические библиотеки для линейной алгебры и статистики

GraphX

- Движок для обработки графов
- Использует RDG (Resilient Distributed Graph)
  - Вершины
  - Ребра
- Использование алгоритма Pregel (supersteps iterations)
- Единый подход как к ETL, так и к обработке графов

> Глоссарий

DAG (directed acyclic graph) - направленный ациклический граф. Каждая вершина в DAG – некоторая операция, которая выполняется над RDD

RDD (Resilient Distributed Dataset) - это распределенный неизменяемых наборов данных на разных узлах кластера, разделенный на один или несколько разделов (partition).

Driver - master-процесс, который преобразует приложения в таски, планирует их запуск на executor-ах и отслеживает прогресс выполнения.

Executors - запущенные java процессы на различных машинах в кластере, выполняющие параллельную обработку данных. Отвечают за выполнение тасок.

SparkContext - абстракция, позволяющая обеспечить единую точку входа для выполнения рапределенных вычислений.

Cluster manager - внешний сервис, который предоставляет доступ к ресурсам кластера (например, standalone manager, YARN, Mesos)

Job - некое параллельное вычисление, состоящее из набора стадий, которые возникают в ответ на action (например, save, count, collect)

Stage (стадия) - одна из частей, на которые разделяется джоба. Стадии зависят друг от друга (аналогично этапам Map и Reduce).

Task - атомарная операция, которая выполняется на executor-е

DAGScheduler - планировщик графов, который разбивает джобы на стейджы и запускает их до готовности.

TaskScheduler - планировщик запуска и мониторинга атомарных процессов обработки данных (тасок).