

Распределённая обработка данных с Apache Spark

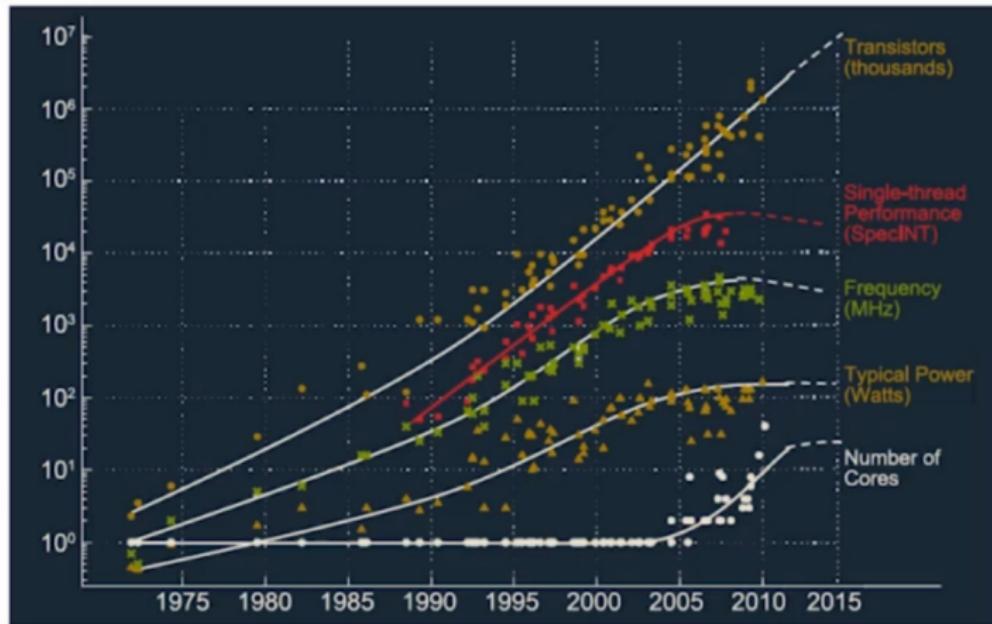
Притыковская Наташа

1 марта 2018 г.

Overview

- 1 Параллельные вычислительные системы
- 2 MapReduce
- 3 Apache Spark

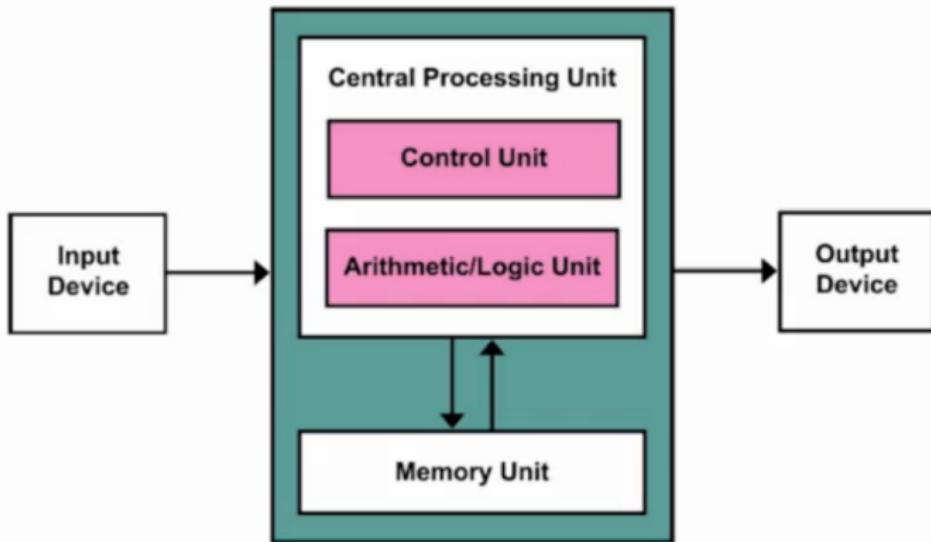
Причина возникновения параллельных вычислений



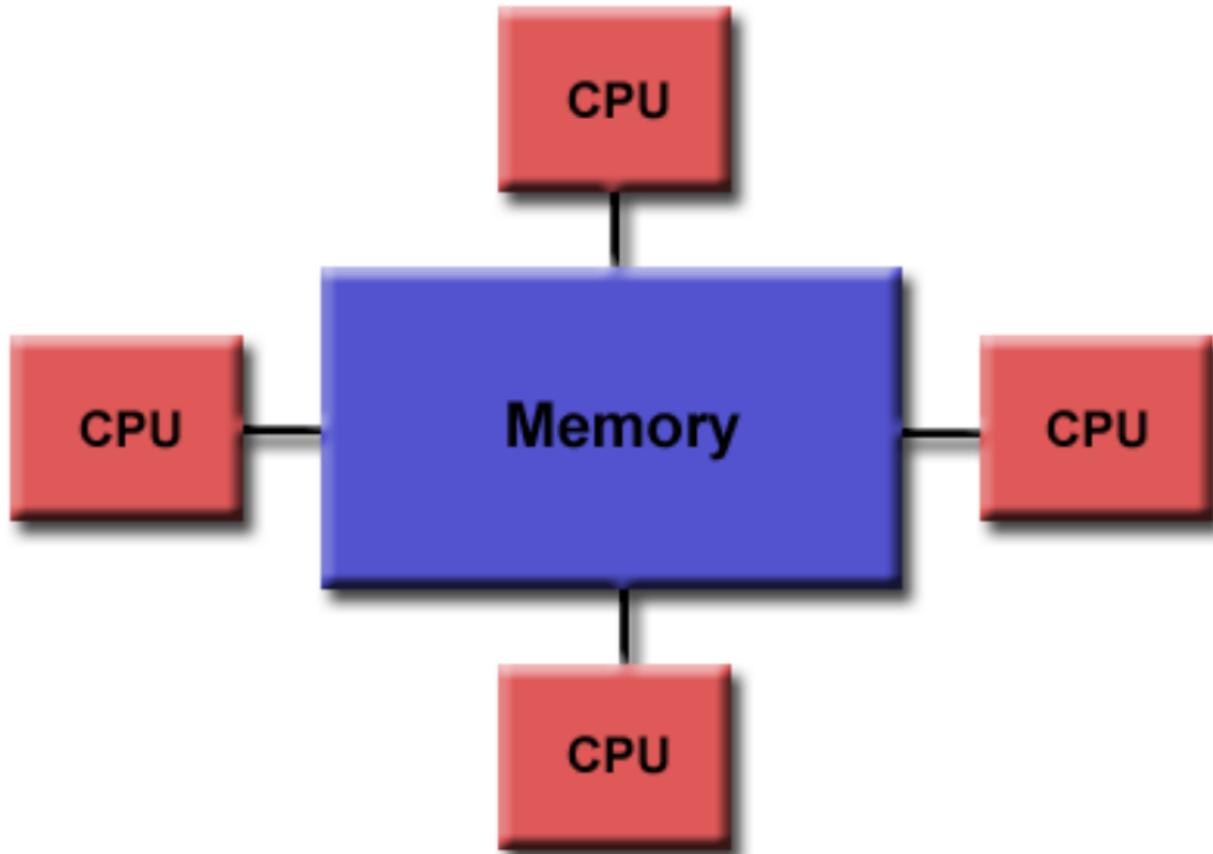
Использование нескольких процессоров для

- Решения задачи за меньшее время
- Решения больших задач, чем на одном процессоре

Архитектура фон Неймана



Общая разделяемая память



Общая разделяемая память

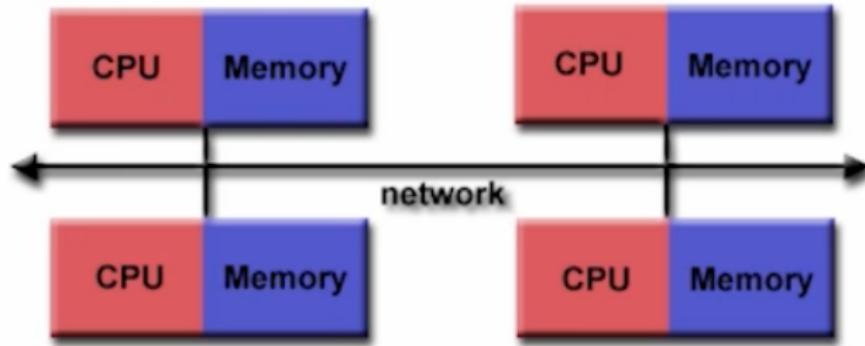
Преимущества

- Привычная модель программирования
- Высокая скорость обмена данными

Недостатки

- Синхронизация при доступе к общим данным
- Масштабируемость

Распределенная память



- Кластера

Преимущества

- Меньшая стоимость
- Высокая масштабируемость

Недостатки

- Необходимость использования сообщений
- Высокая латентность и низкая пропускная способность => оптимизация распределения данных между процессорами

Характеристики сети

- Латентность L (latency) время между началом отправки данных на отправляющей стороне и получением первого байта на принимающей стороне
- Пропускная способность B (bandwidth) скорость, с которой данные получает принимающая сторона
- Время передачи сообщения $T = L + \frac{N}{B}$, где N - размер сообщения

1990



2000



2010



Максимально достижимое ускорение

- доля последовательных вычислений

$$f = \frac{T_{seq}}{T_1}$$

- время выполнения параллельного алгоритма

$$T_p = fT_1 + \frac{(1-f)T_1}{p}$$

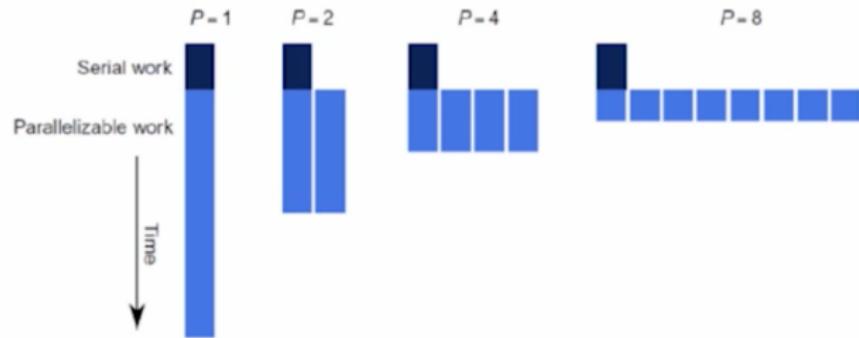
- Ускорение

$$S_p = \frac{1}{f + \frac{1-f}{p}}$$

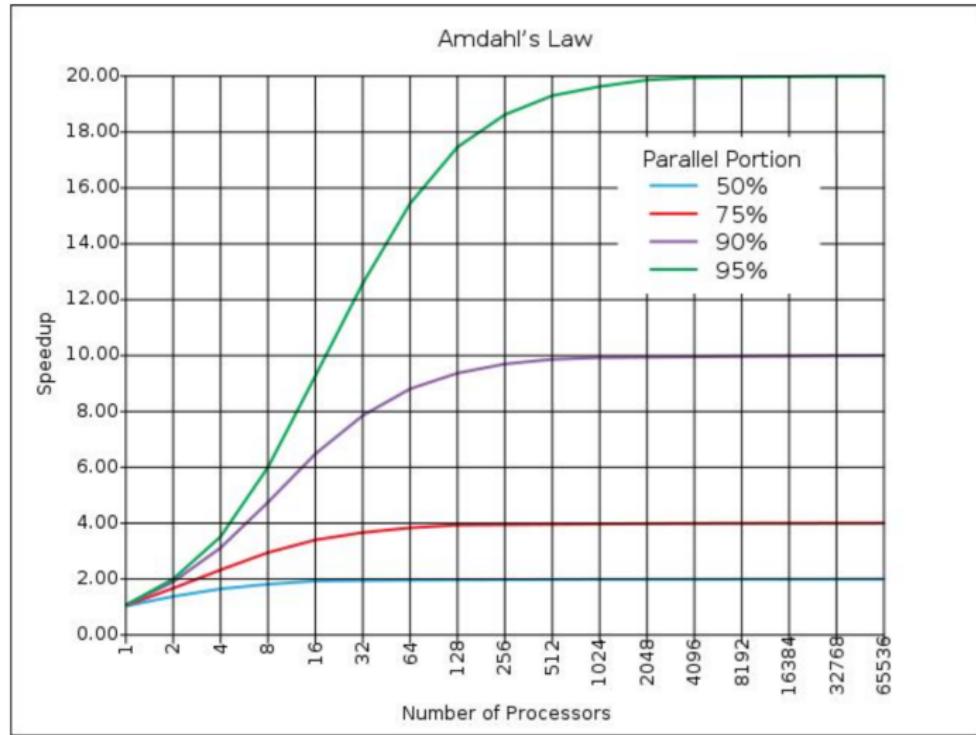
- Максимальное ускорение

$$\lim_{p \rightarrow \infty} S_p = \frac{1}{f}$$

Пример



Пример



Прежде, чем использовать параллеленое программирование

- Стоит ли задача усилий?
- Какие части задачи наиболее интенсивны в вычислительном отношении?
- Есть ли там параллелизм?
- Есть ли готовые параллельные реализации?

MapReduce

- Модель программирования для описания больших объемов данных

Феномен Big Data

- Коммерческие приложения
 - Web
 - миллиарды страниц
 - Google MapReduce: 20 PB / день (2008)
 - Социальные сети
 - Facebook: 600 TB / день (2014)
- Научные приложения
 - Физика высоких энергий
 - Большой Адронный Коллайдер - 15 PB / год (2017)
 - Биоинформатика
 - Секвенирование ДНК, European Bioinformatics Institute

Наблюдения

- Современные задачи намного превышают возможности одной машины - требуются кластера из сотен и тысяч машин
- Данные нельзя полностью разместить в памяти, приходится обращаться к диску - последовательные чтение и запись данных при обработке гораздо эффективнее случайного доступа
- Отказы становятся нормой - необходимы автоматическая обработка и восстановление после отказов
- Разрабатывать приложения с 0 сложно - требуются высокоуровневые модели программирования, скрывающие детали системного уровня

Подсчет частоты встречаемости слов

- Исходные данные: $[(docid, content), \dots]$
- Требуемый результат: $[(term, total_{tf})\dots]$
- Этап 1 - Параллельная обработка документов
 - $(docid, content) \rightarrow [(term1, tf1), (term2, tf2), \dots]$
- Этап 2 - Параллельная агрегация промежуточных результатов для каждого терма
 - $(term, [tf1, tf2, \dots]) \rightarrow (term, total_{tf})$

Модель программирования MapReduce

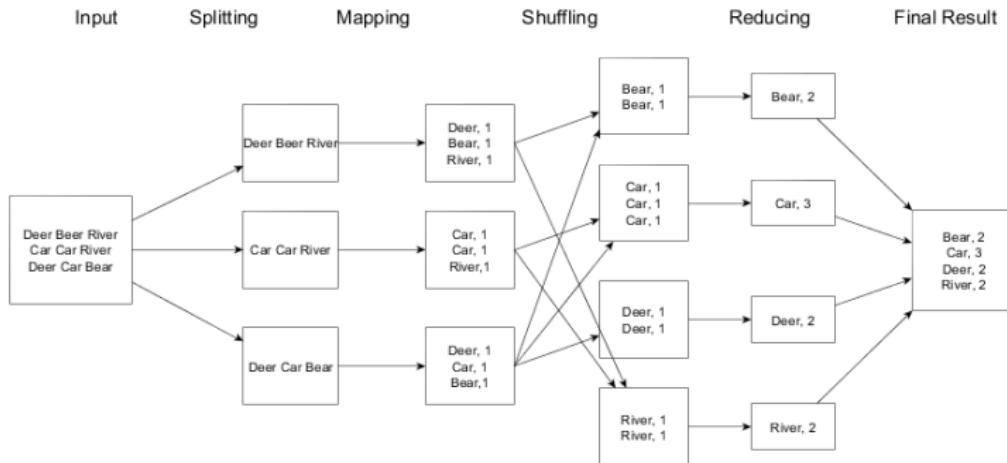
- Базовой структурой данных являются пары (ключ, значение)
- Программа описывается путем определения функций

$$map : (k1, v1) \rightarrow (k2, v2)$$

$$reduce : (k2, [v2]) \rightarrow (k3, v3)$$

- Между map и reduce происходит группировка и сортировка промежуточных данных по ключу k2

Подсчет частоты встречаемости слов



Запуск MapReduce-программы

- Конфигурация задания
 - Входные данные
 - Функции map, reduce
 - Местоположение и формат выходных данных

Обязанности реализации

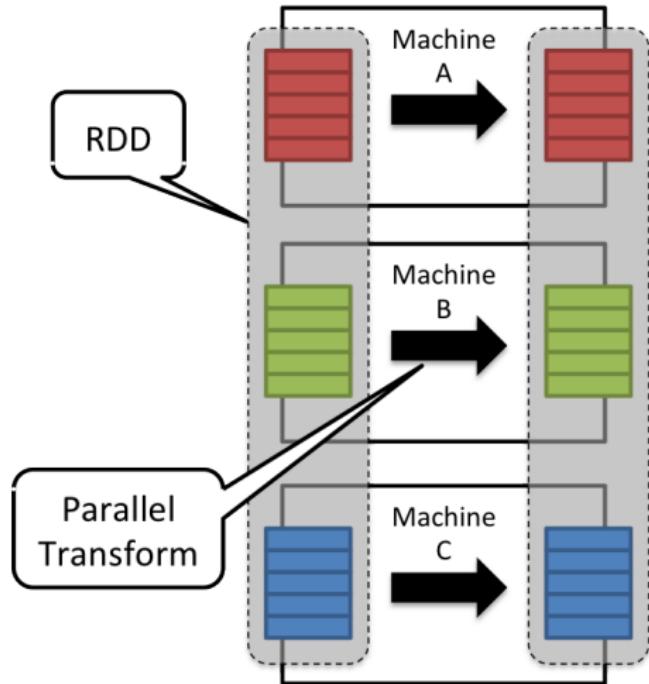
- Декомпозиция на параллельные подзадачи (map- и reduce-задачи)
- Запуск рабочих процессов
- Распределение задач по рабочим процессам и балансировка нагрузки
- Передача данных рабочим процессам (требуется минимизировать)
- Синхронизация и передача данных между рабочими процессами
- Обработка отказов рабочих процессов



Resilient Distributed Dataset (RDD)

- Данные, которыми оперирует программа Spark
- Распределенные коллекции
- Не поддерживают модификацию
- Могут быть созданы путем
 - использования существующей коллекции
 - чтения данных из внешнего хранилища (например из HDFS)
 - трансформации существующих RDD

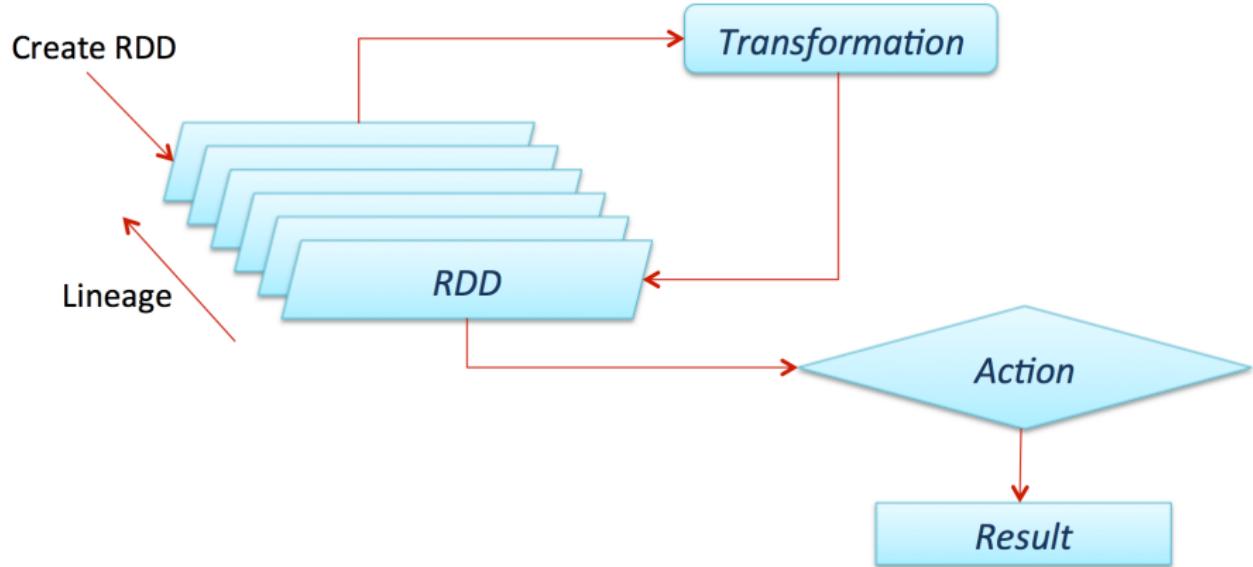
RDD



Операции над RDD

- Трансформации (возвращают RDD)
 - map, filter, flatMap, sample
 - groupByKey, reduceByKey, sortByKey, join
- Действия (action): возвращают вычисленное значение или сохраняют RDD
 - count, first, take, takeSample, foreach, countByKey
 - collect, saveAsTextFile
- Кеширование данных
 - cache

RDD



WordCount на Spark

- Scala API

```
val lines = spark.textFile("/data/docs")
val counts = lines.flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)
counts.saveAsTextFile("counts")
```

- Python API

```
lines = spark.textFile("/data/docs")
counts = lines.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("counts")
```

TopWords (1)

```
from __future__ import print_function
import sys
from operator import add
from pyspark import SparkContext

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: topwords <input> <output>", file=sys.stderr)
        exit(-1)

    input_path = sys.argv[1]
    output_path = sys.argv[2]

    # setup spark context
    sc = SparkContext(appName="Top Words")

    # read input data
    lines = sc.textFile(input_path)
```

TopWords (2)

```
# split to words, perform word count and cache results
counts = lines.flatMap(lambda line: line.lower().split(' '))
    .map(lambda word: (word, 1))
    .reduceByKey(add)
    .cache()

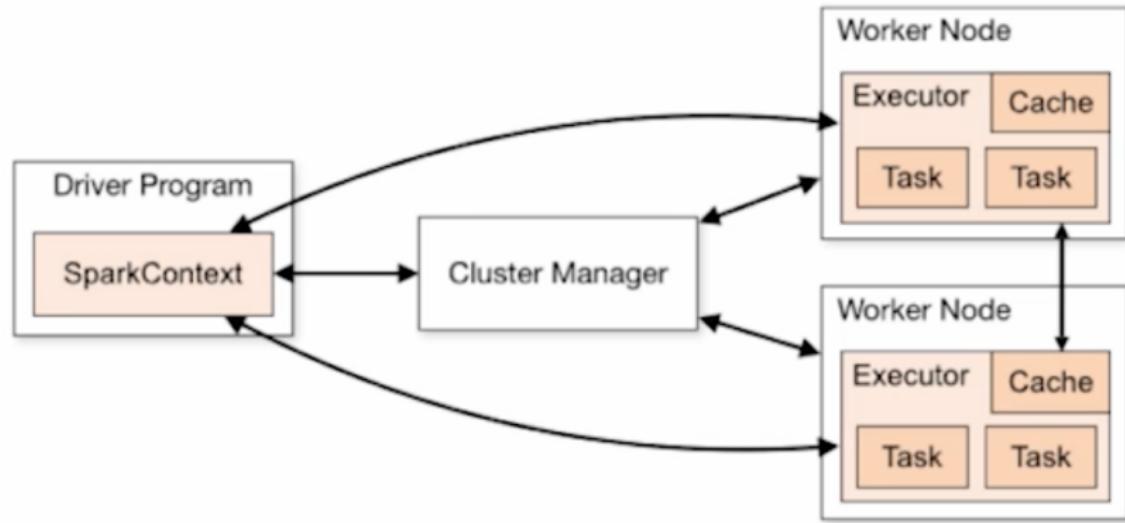
# convert results to convenient TSV format and save
counts.map(
    lambda (word, count): u'%s\t%d' % (word, count)
).saveAsTextFile(output_path + '-counts')

# compute top 20 most frequent words
top_words = counts.filter(lambda (word, count): count > 100)
    .sortBy(lambda pair: pair[1], ascending=False)
    .take(20)
```

Режимы выполнения программы

- Распределенный режим (cluster)
 - Выполнение программы на узлах кластера (Hadoop YARN, Apache Mesos)
 - Работа с данными в распределенном хранилище (HDFS, Cassandra)
- Локальный режим (local)
 - Выполнение программы на локальной машине
 - Работа с данными в локальной файловой системе
 - Предназначен для изучения Spark, разработки и отладки программ

Распределенный режим выполнения



Распределенный режим выполнения YARN

