Технологии работы с большими данными

Лекция 5. Hadoop & Spark. Теория

нетология

Алексей Кузьмин

Директор разработки; ДомКлик.ру

О спикере:

- Руковожу направлением работы с данными и Data Science
- Преподаю в Нетологии
- Работаю в IT с 2010 года (ABBYY, ДомКлик)
- Окончил МехМат МГУ в 2012 году

Я в Слаке:

@Alexey Kuzmin

Работа с данными

Источники данных

Сбор данных
Лекция 8

SQL-БД
Лекция 1

NoSQL
Лекция 4

MapReduce
Лекция 5-7

Мотивация Big Data
Лекция 3

Data Science
Лекция 2

Отчеты
Лекция 1

Модели
Лекция 2

Люди и процессы
Лекция 9

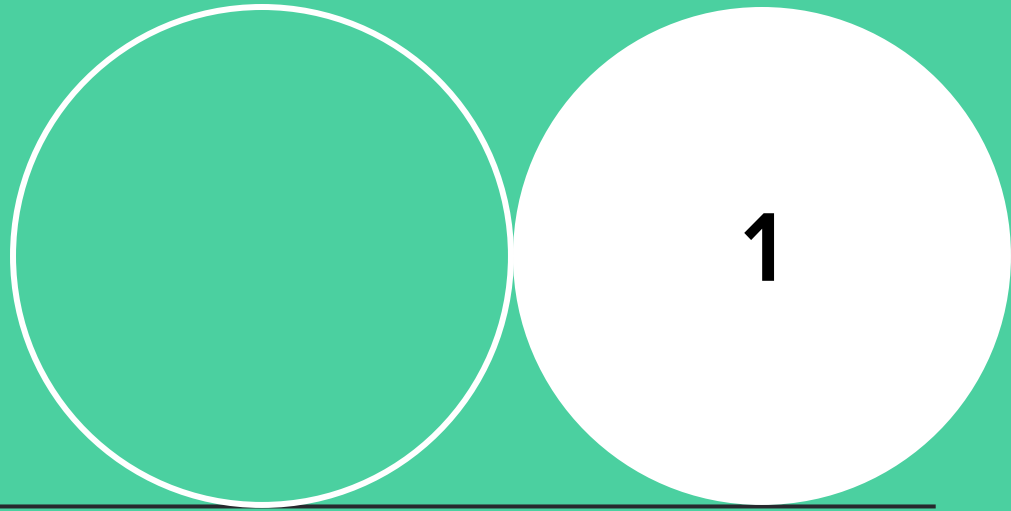
Примеры кейсов
Лекция 10

План

- Познакомимся с основными инструментами обработки больших данных и лежащих под ними принципами
- Сегодня - теоретическая часть
- Следующие две лекции - практика

Hadoop

Пионер больших данных



Алексей Кузьмин

Технологии работы с большими данными

нетология

Hadoop

Hadoop — проект фонда Apache Software Foundation, свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределённых программ, работающих на кластерах из сотен и тысяч узлов.

https://ru.wikipedia.org/wiki/Hadoop

История

- Разработка начата в 2005-ом году
- Базируется на идеях Google о вычислительной концепции Map&Reduce
- Начиная с 2009 начинает активно использоваться крупными компаниями для работы с большими данными

Основные компоненты

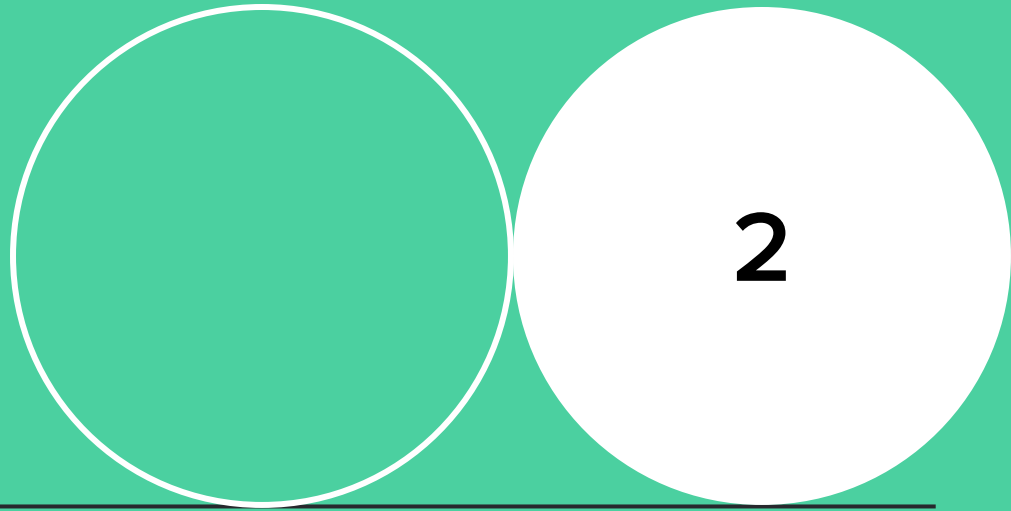
- HDFS - распределенная файловая система, отвечающая за хранение данных
- MapReduce - вычислительная часть, позволяющая обрабатывать данные в парадигме map & reduce
- Yarn - модуль управления ресурсами кластера (появился в версии 2.0). Не будем рассматривать

Кластер Hadoop

- “Дешевое” обычное железо (Commodity Hardware), соединенное по сети
- Горизонтальное масштабирование вместо вертикального
- Автоматическая обработка падений и отказов оборудования
- Инкапсуляция сложности работы распределенных и многопоточных приложений

HDFS

Hadoop Distributed File System



Алексей Кузьмин

Технологии работы с большими данными

нетология

HDFS

- *Hadoop Distributed File System*
- Для пользователя как “*один большой диск*”
- Работает поверх обычных файловых систем
- Основывается на архитектуре *Google’s Filesystem GFS research.google.com/archive/gfs-sosp2003.pdf*
- Fault Tolerant – Умеет справляться с выходом из строя дисков, машин и т.д.

HDFS хорош для

- Хранения больших файлов
  - Терабайты, петабайты..
  - Миллионы (но не миллиарды) файлов – Файлы размером от 100 Мб
- Стриминг данных
  - Паттерн “write once / read-many times”
  - Оптимизация под последовательное чтение
    - Нет операциям произвольного чтения
- Обычные сервера

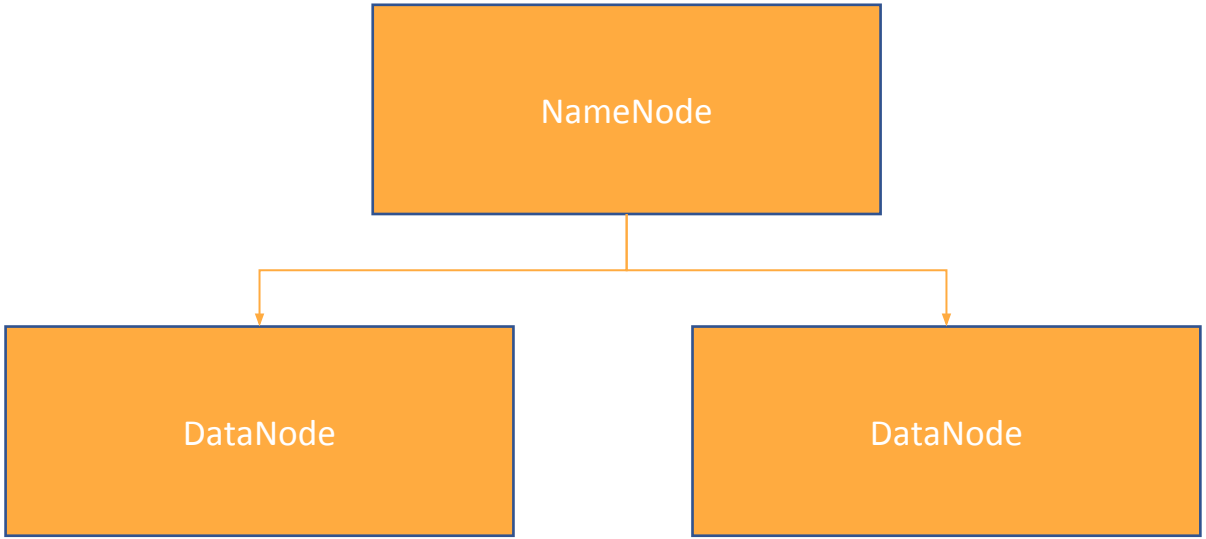
Файлы в HDFS

- Файлы в HDFS состоят из блоков
  - Блок – единица хранения данных
  - Стандартный размер блоков 64Мб или 128Мб
- Реплицируются по машинам в процессе записи
  - Один и тот же блок хранится на нескольких Datanode
- Фактор репликации по умолчанию равен **3**

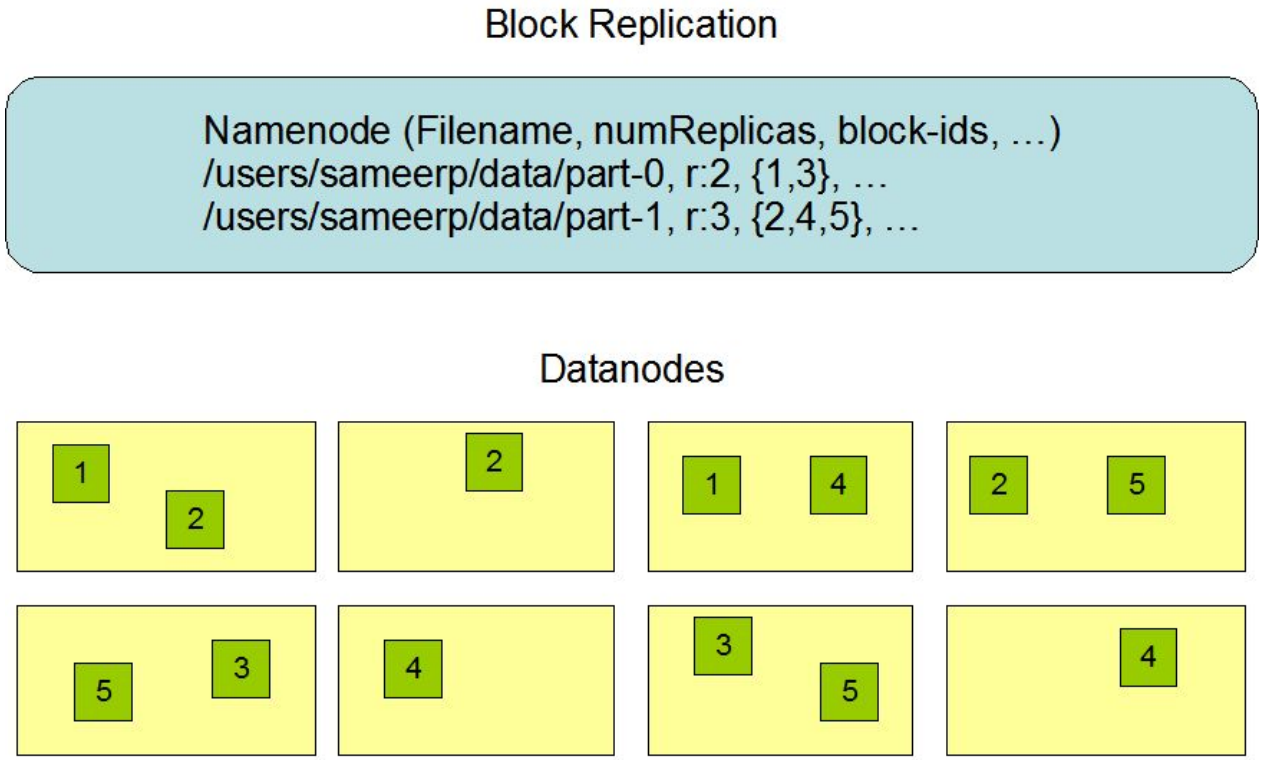
Узлы HDFS

- Для управления файловой системой есть два основных типа узлов (компьютеров)
- Namenode:
  - Отвечает за файловое пространство (namespace), мета-информацию
    - Управляет расположением блоков файлов
    - Запускается на выделенной машине (иногда добавляют secondary namenode)
- Datanode
  - Хранит и отдает блоки данных
  - Отправляет ответы о состоянии на Namenode
  - Запускается обычно на всех машинах кластера

Узлы HDFS



Файлы в HDFS


Block Replication
Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

Репликация блоков

- Namenode определяет, куда копировать реплики блоков
- Размещение блоков зависит от того, в какой стойке стоит сервер
  - Баланс между надежностью и производительностью
  - Попытка снизить нагрузку на сеть (*bandwidth*)
  - Попытка улучшить надежность путем размещения реплик в разных стойках
  - Фактор репликации по умолчанию равен 3
    - 1-я реплика на локальную машину
    - 2-я реплика на другую машину из той же стойки
    - 3-я реплика на машину из другой стойки

Принцип работы

- Namenode не выполняет непосредственно операций чтения/записи данных
  - Это одна из причин масштабируемости Hadoop
- Клиент обращается к Namenode для получения информации о размещении блоков для чтения/записи
- Клиент взаимодействует напрямую с Datanode для чтения/записи данных

Принцип работы

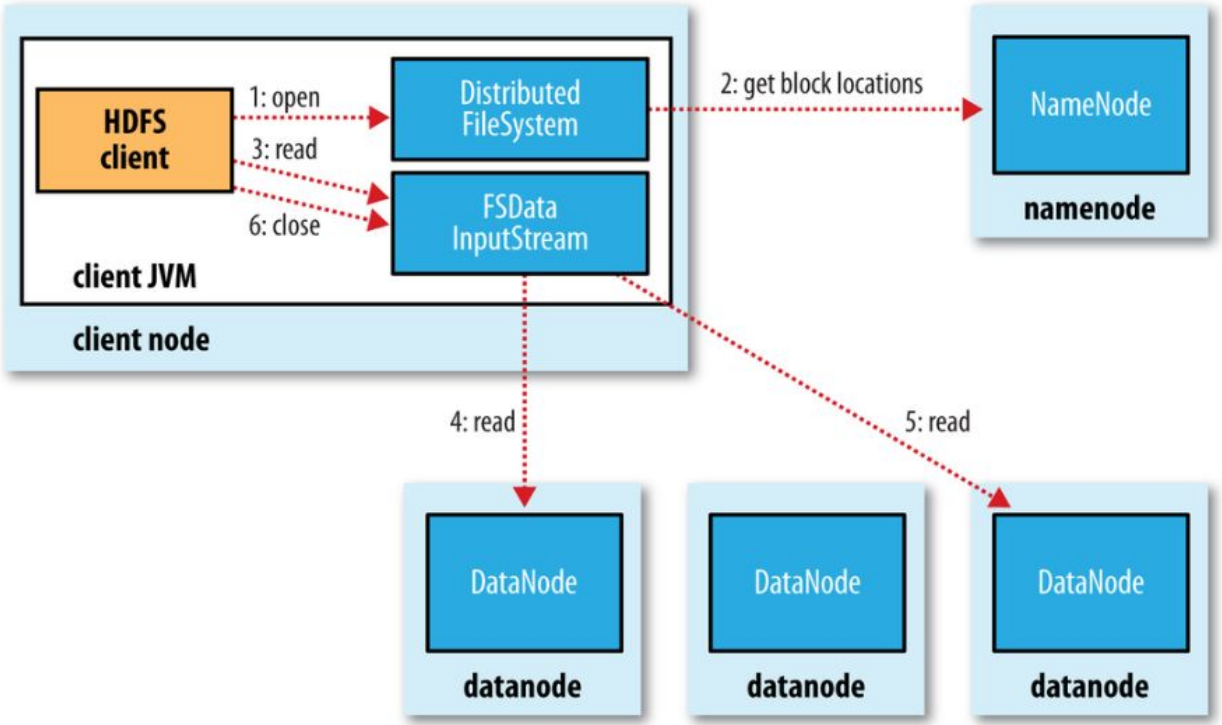
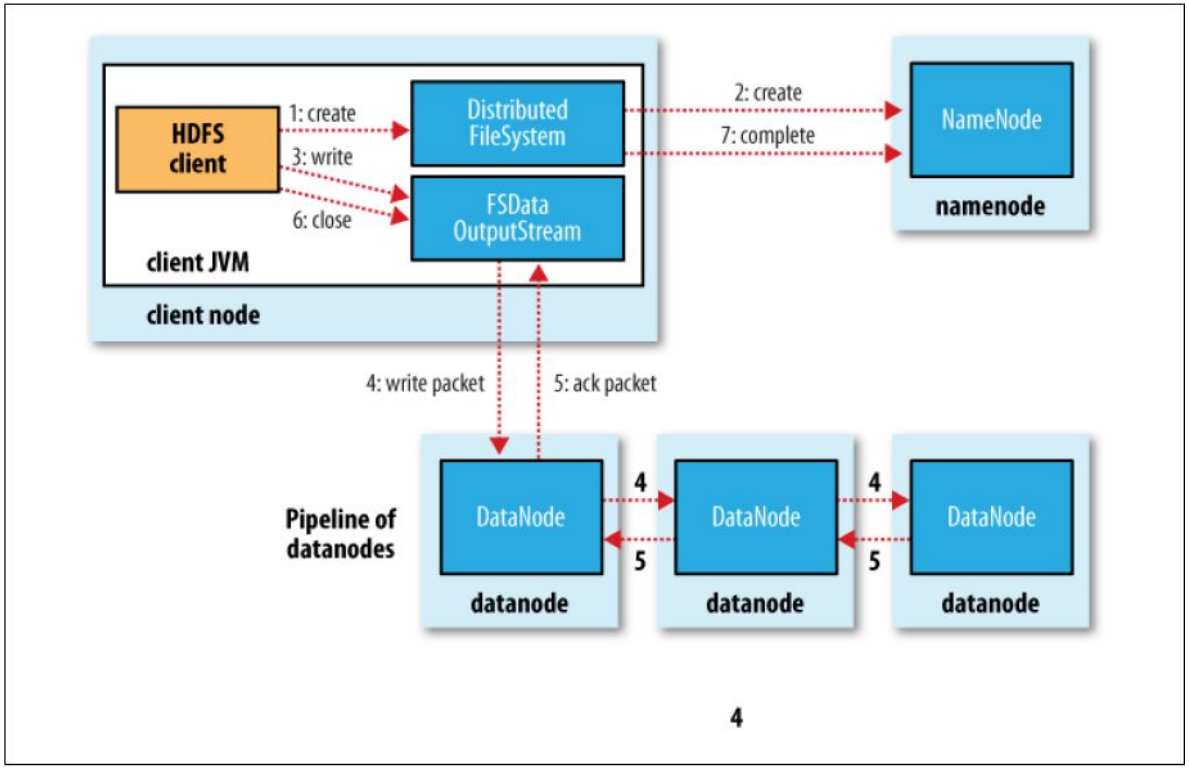


Figure 3-2. A client reading data from HDFS

Запись файла

1. Запрос на запись
2. Создаем файл на NN и определяем расположение блоков
3. Начинаем запись
4. Записываем файлы по все ноды
5. Получаем подтверждение
6. Закрываем файл
7. Отправлем подтверждение на NN

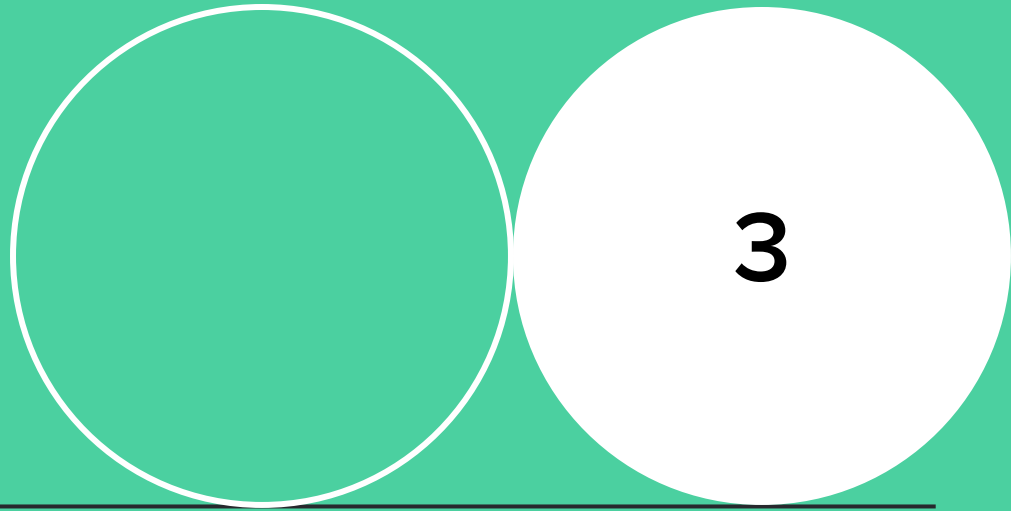


NameNode

- Для быстрого доступа вся мета-информация о блоках хранится в ОЗУ Namenode
  - Чем больше кластер, тем больше ОЗУ требуется
  - Лучше миллионы больших файлов (сотни мегабайт), чем миллиарды маленьких
  - Работает на кластерах из сотен машин
- Hadoop 2+
  - Namenode Federation
  - Каждая Namenode управляет частью блоков
  - Горизонтальное масштабирование Namenode
  - Поддержка кластеров из тысячи машин

MapReduce

Обрабатываем данные



Алексей Кузьмин

Технологии работы с большими данными

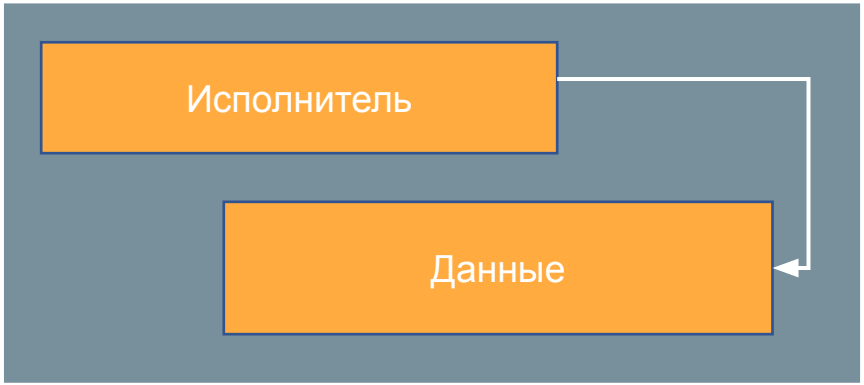
нетология

Доставка кода к данным

- Традиционная архитектура системы обработки данных
  - Узлы системы разделяются на вычислительные и хранилища, соединенные по сети

Доставка кода к данным

- Hadoop сближает вычислительные узлы и данные
  - Код копируется к данным (небольшой расход, Кб)
  - Исполнитель выполняет код на локально расположенных данных



Узел Hadoop

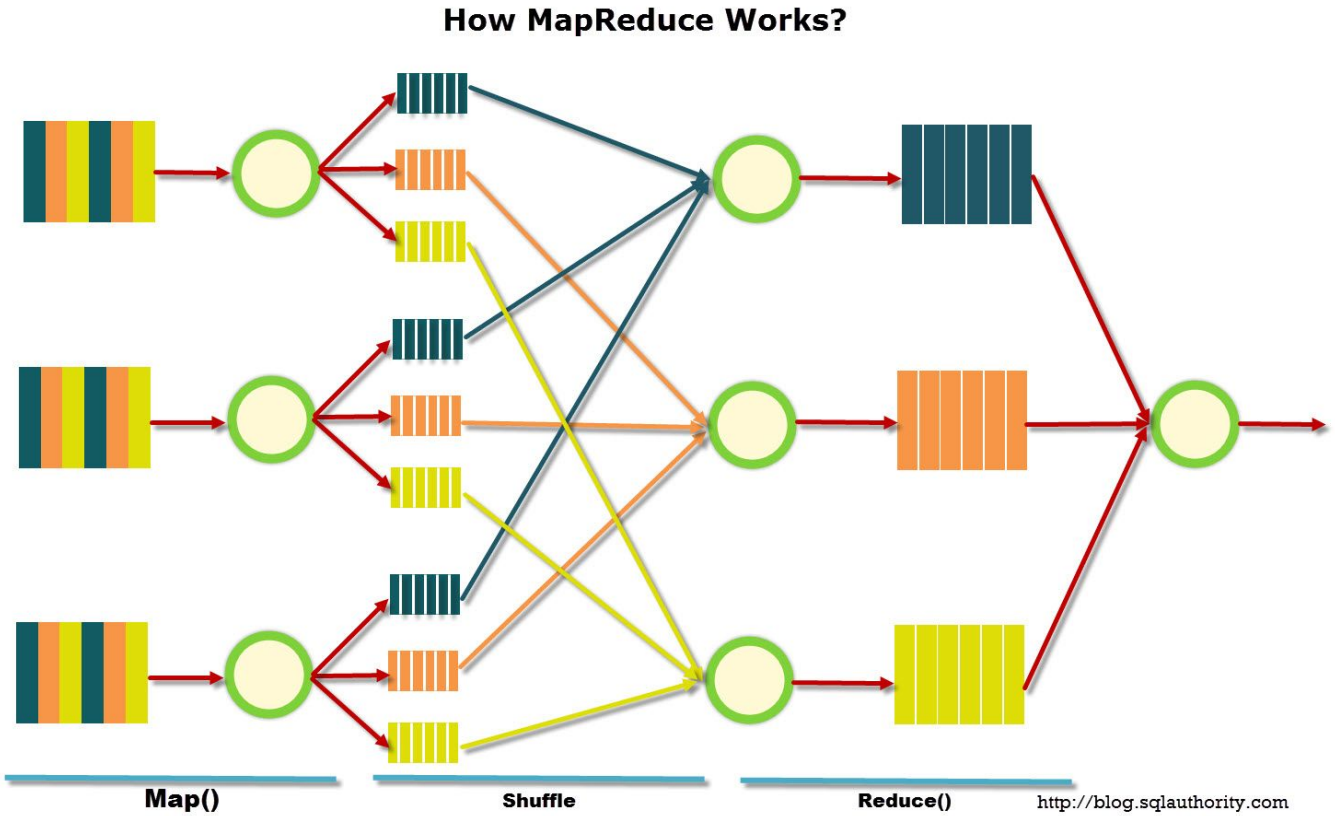
MapReduce

- Самая известная парадигма обработки больших данных
- Предложена компанией Google в 2004 году
- Имеет множество имплементаций (в том числе open source)
- Основная идея - принцип “Разделяй и властвуй”
- 3 шага - map, shuffle и reduce

MapReduce

- **Map-шаг** - пред обработать данные и выделить в них ключ - признак по которому данные будут агрегированы
    - Вход – исходный объект
    - Выход – множество пар ключ-значение
- **Shuffle-шаг** – разложить данные по “корзинам” в соответствии с ключом
- **Reduce-шаг** - данные обладающие одним ключом обрабатываются вместе.
    - Вход – Ключ -> список значений
    - Выход – Ключ –> значение

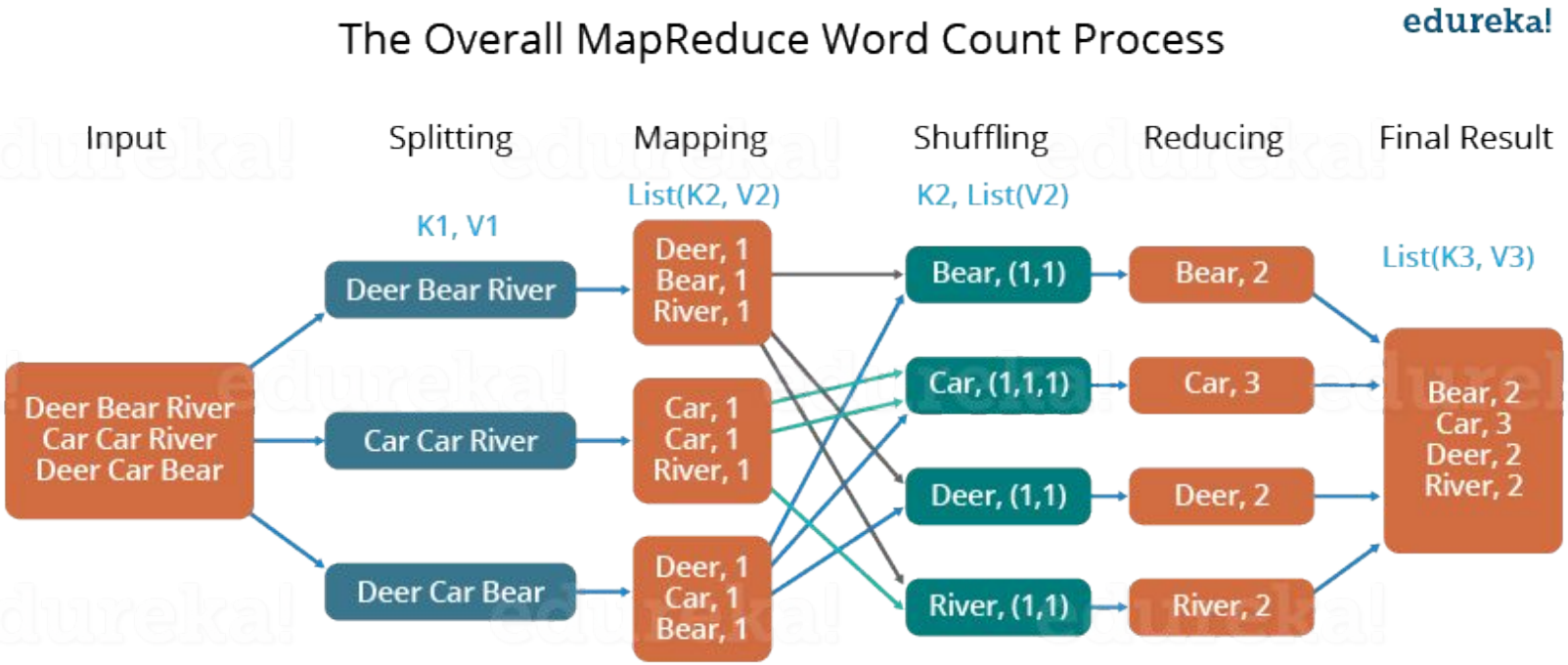
Схема работы



Word Count

- Дано – файл со строками.
- Посчитать:
  - Сколько раз встречается каждое слово в исходном файле

WordCount

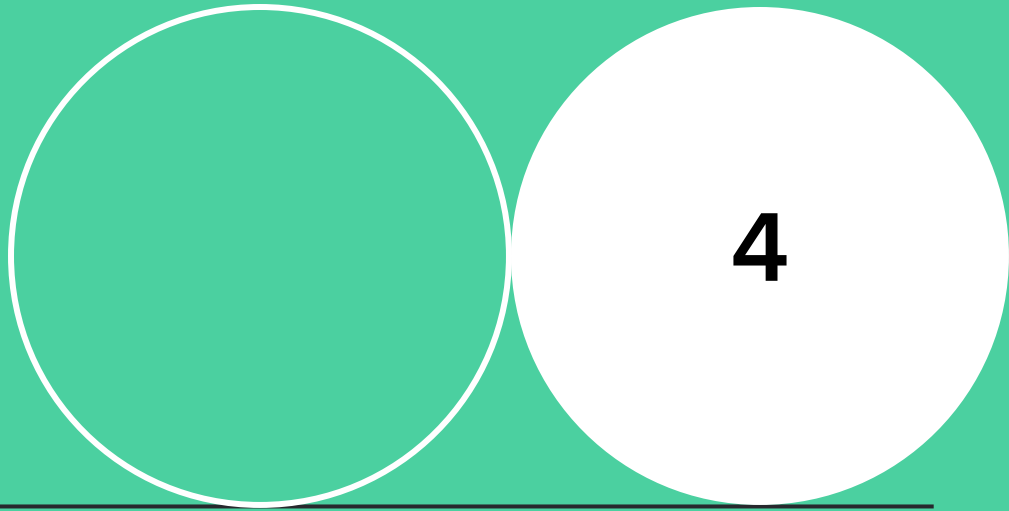

The Overall MapReduce Word Count Process
edureka!
Input
Splitting
Mapping
Shuffling
Reducing
Final Result
K1, V1
List(K2, V2)
K2, List(V2)
List(K3, V3)
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, (1,1)
Car, (1,1,1)
Deer, (1,1)
River, (1,1)
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

Python

```
def map(string):
    for word in string.split():
        yield (word , 1)

def reduce(word, list_of_one):
    return (word, sum(list_of_one)
```

Shuffle шаг реализуется фреймворком и не задается пользователем

Соберем все вместе

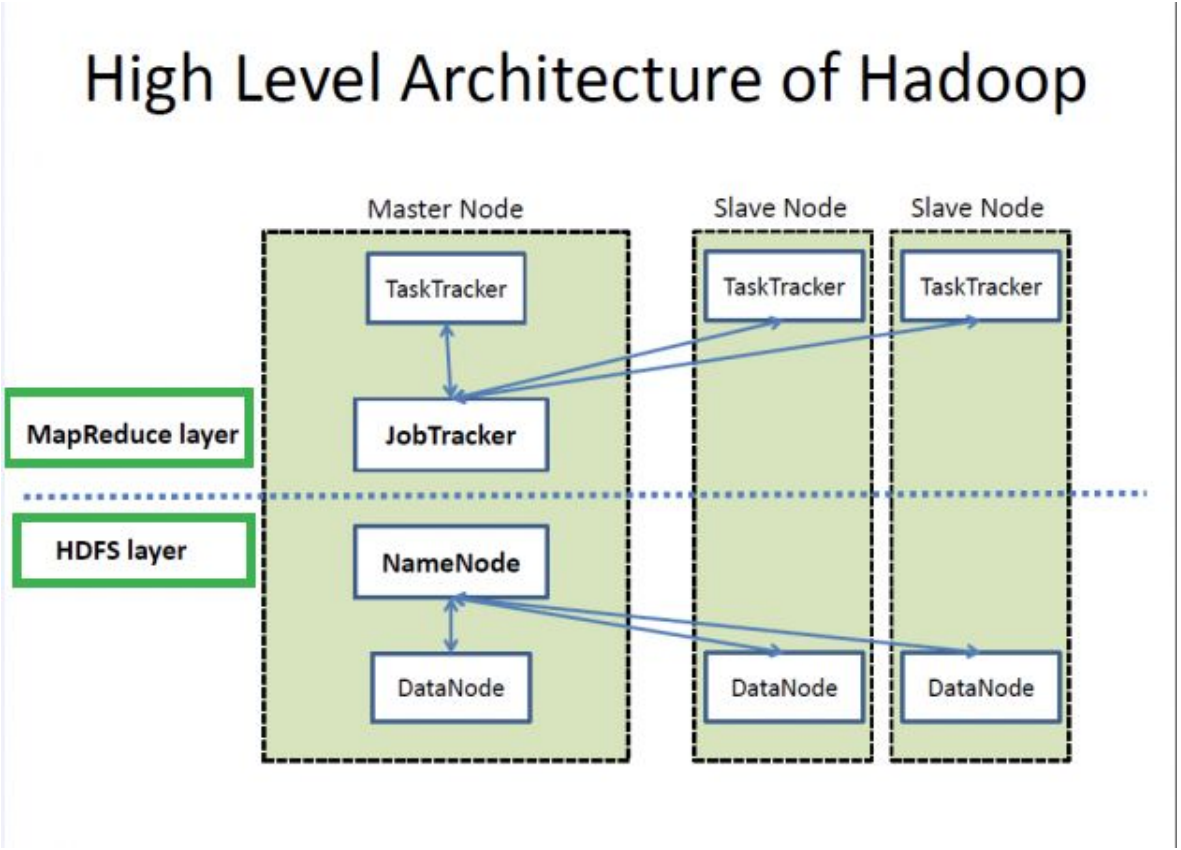


Алексей Кузьмин

Технологии работы с большими данными

нетология

MapReduce & HDFS



MapReduce “Runtime”

- Управление запуском задач
  - Назначает воркерам таски *map* или *reduce*
- Управление “*data distribution*” – перемещает код к данным и запускает задачи по возможности локально с данными
- Управление синхронизацией
  - Собирает, сортирует и объединяет промежуточные данные
- Управление обработкой ошибкой и отказов
  - Определяет отказ воркера и перезапускает таск
- Все работает поверх распределенной FS

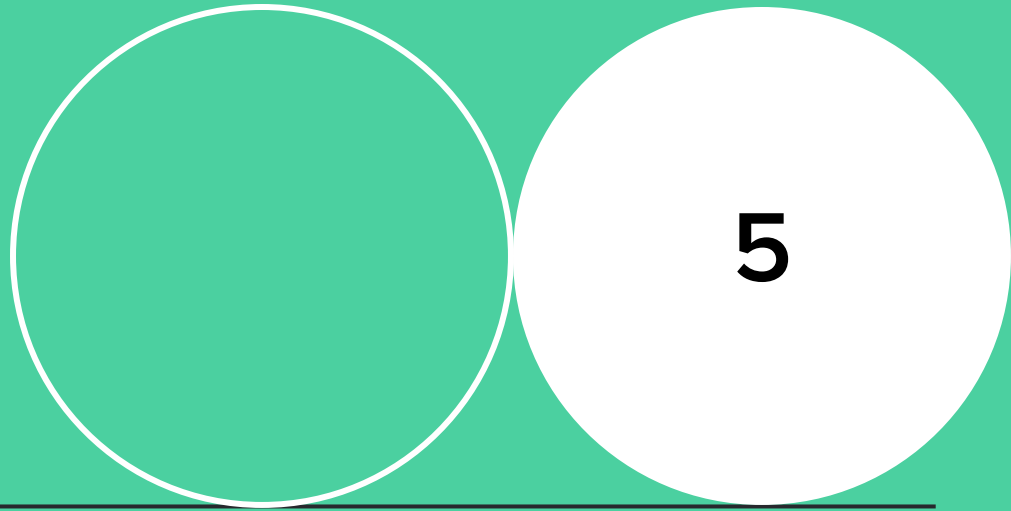
MapReduce

- MapReduce 1.0 работает поверх демонов
  - ***JobTracker*** и ***TaskTracker***
- ***JobTracker***
  - Управляет запуском тасков и определяет, на каком ***TaskTracker*** таск будет запущен
  - Управляет процессом работы MapReduce задач (*jobs*)
  - Мониторит прогресс выполнения задач
  - Перезапускает зафейленные или медленные таски
- MapReduce имеет систему ресурсов основанную на слотах (*slots*)
  - На каждом ***TaskTracker*** определяется, сколько будет запущено слотов
  - Таск запускается в одном слоте
  - M мапперов + R редьюсеров = N слотов

Еще в Hadoop

- **Hadoop YARN** – фреймворк для управления ресурсами кластера и менеджмента задач, в том числе включает фреймворк MapReduce,
- **Hive** – инструмент для SQL-like запросов над большими данными (превращает SQL-запросы в серию MapReduce–задач);
- **Pig** – язык программирования для анализа данных на высоком уровне. Одна строчка кода на этом языке может превратиться в последовательность MapReduce-задач;
- **Hbase** – колоночная база данных, реализующая парадигму BigTable;
- **Cassandra** – высокопроизводительная распределенная база данных;
- **ZooKeeper** – сервис для распределённого хранения конфигурации и синхронизации изменений этой конфигурации;
- **Mahout** – библиотека и движок машинного обучения на больших данных.

Spark



Алексей Кузьмин

Технологии работы с большими данными

нетология

Мотивация

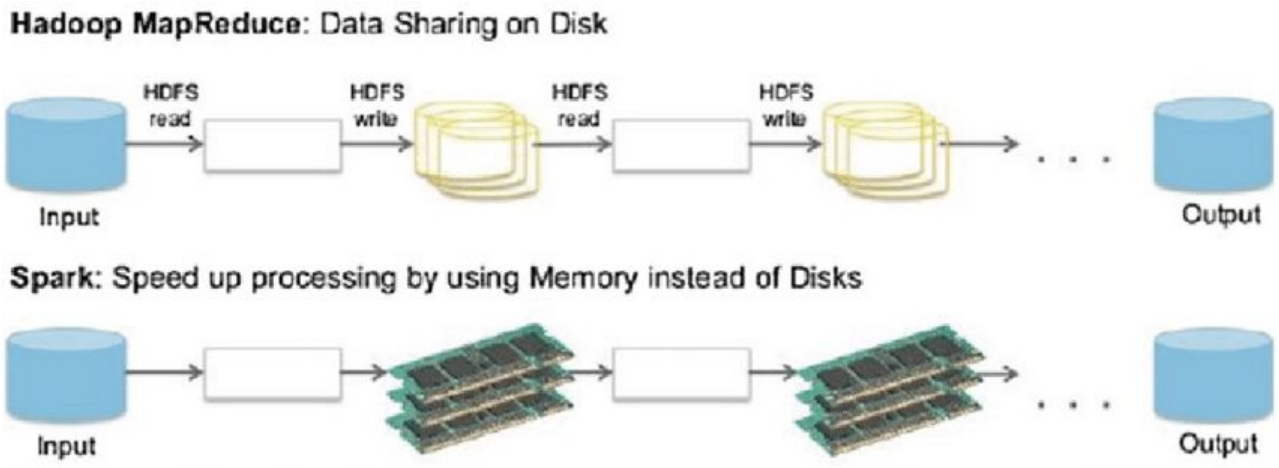
- *MapReduce* отлично упрощает анализ ***big data*** на больших, но ненадежных, кластерах
- Но с ростом популярности фреймворка пользователи хотят большего:
  - **Итеративных** задач, например, алгоритмы machine learning
  - **Интерактивной** аналитики

Мотивация

- Для решения обоих типов проблем требуются одна вещь, которой нет в MapReduce...
  - Эффективных примитивов для общих данных (*Efficient primitives for data sharing*)
- В MapReduce единственный способ для обмена данными между задачами (*jobs*), это надежное хранилище (*stable storage*)
- Репликация также замедляет систему ,но это необходимо для обеспечения *fault tolerance*

Решение

- Обработка и разделение данных в памяти (RAM)



Resilient Distributed Datasets (RDD)

- Задача - Разработать дизайн абстракции распределенной памяти с поддержкой **fault tolerant** и **эффективности**
- Абстрактное представление распределенной RAM
- Immutable коллекция объектов распределенных по всему кластеру
- RDD делится на партиции, которые являются атомарными частями информации
- Партиции RDD могут хранится на различных нодах кластера

Программная модель

- Основана на RDD
- С которыми можно выполнять 2 вида операций:
  - Трансформации (Transformations)
  - Действия (Actions)

Трансформации

Результатом применения к RDD является новый RDD. Это операции, которые каким-либо образом преобразовывают элементы. Примеры:

- **.map(function)** — применяет функцию function к каждому элементу датасета
- **.filter(function)** — возвращает все элементы датасета, на которых функция function вернула истинное значение
- **.distinct([numTasks])** — возвращает датасет, который содержит уникальные элементы исходного датасета
- и другие

Действия

Действия применяются тогда, когда необходимо получить результат — сохранить данные на диск, либо вывести часть данных в консоль. Примеры:

- **.saveAsTextFile(path)** — сохраняет данные в текстовый файл (в hdfs, на локальную машину или в любую другую поддерживаемую файловую систему — полный список можно посмотреть в документации)
- **.collect()** — возвращает элементы датасета в виде массива. Как правило, это применяется в случаях, когда данных в датасете уже мало (применены различные фильтры и преобразования) — и необходима визуализация, либо дополнительный анализ данных, например средствами пакета Pandas
- **.take(n)** — возвращает в виде массива первые n элементов датасета
- **.count()** — возвращает количество элементов в датасете

Операторы

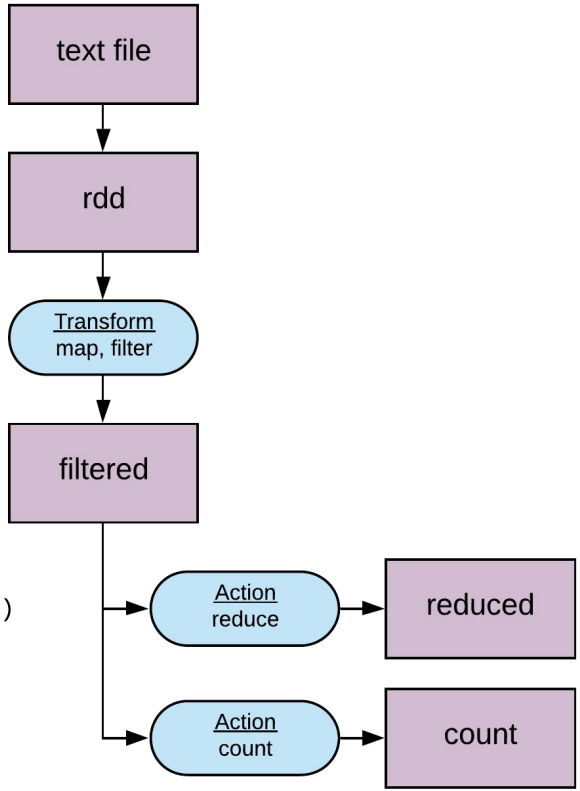
Transformations	*map*(f : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(f : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(f : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(f : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(f : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(c : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(p : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(f : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(k : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

Программная модель

- RDD проходят через череду трансформаций
  - Трансформации применяются отложено (lazy)
  - То есть вычисления происходят непосредственно в момент получения результата, а не заранее
- Затем получается результат при помощи действия
  - Запускается вычисление всех участвующих трансформаций
  - Результат выводится на консоль, сохраняется в файл и тп
- Набор трансформаций и действий с данными можно представить в виде графа (DAG - direct acyclick graph)

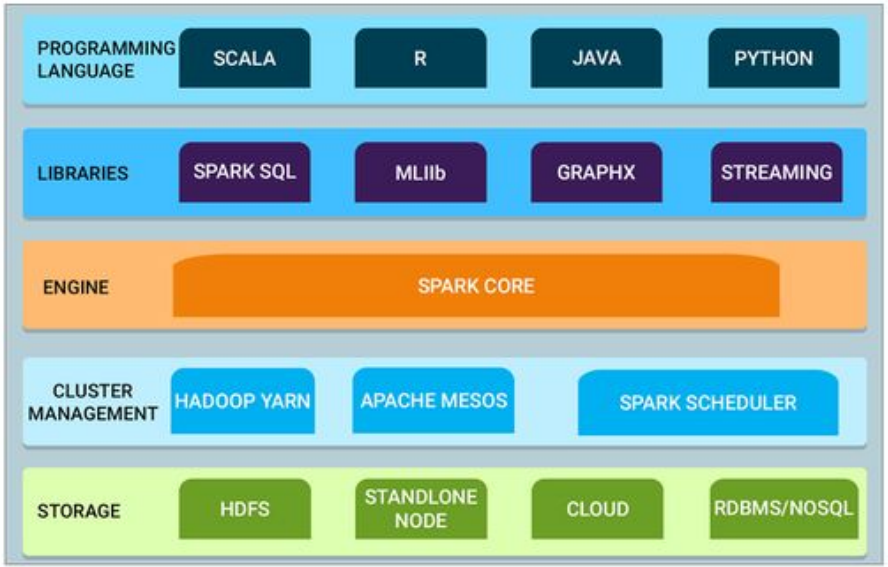
Пример DAG'a

```
val rdd = sc.textFile(...)

val filtered = rdd.map(...)
                  .filter(...)
                  .persist()

val reduced = filtered.reduce(...)

val count = filtered.count()
```



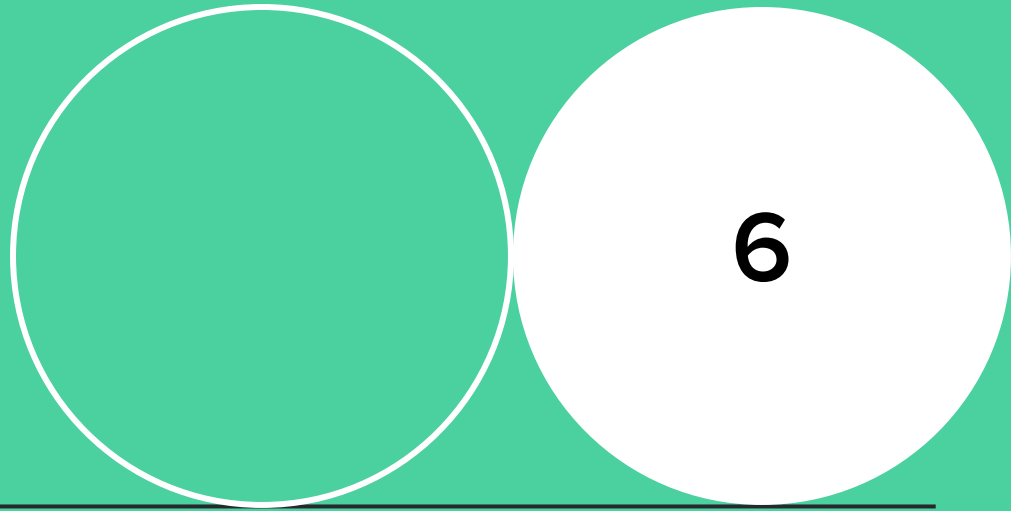
Еще в Spark

- **MLLib** – библиотека машинного обучения для spark.
- **graphx** – библиотека работы с графами для apache spark
- **SparkSQL** – библиотека для трансляции SQL-запросов (альтернатива HIVE)
- **Spark streaming** – библиотека обработки потоковых данных



Демо для желающих

Пример работы Hadoop



Алексей Кузьмин

Технологии работы с большими данными

нетология

Запуск

Как развернуть локально?

Проще всего - запустить в докере.

Инструкция - https://hortonworks.com/tutorial/sandbox-deployment-and-install-guide/section/3/

Начало работы

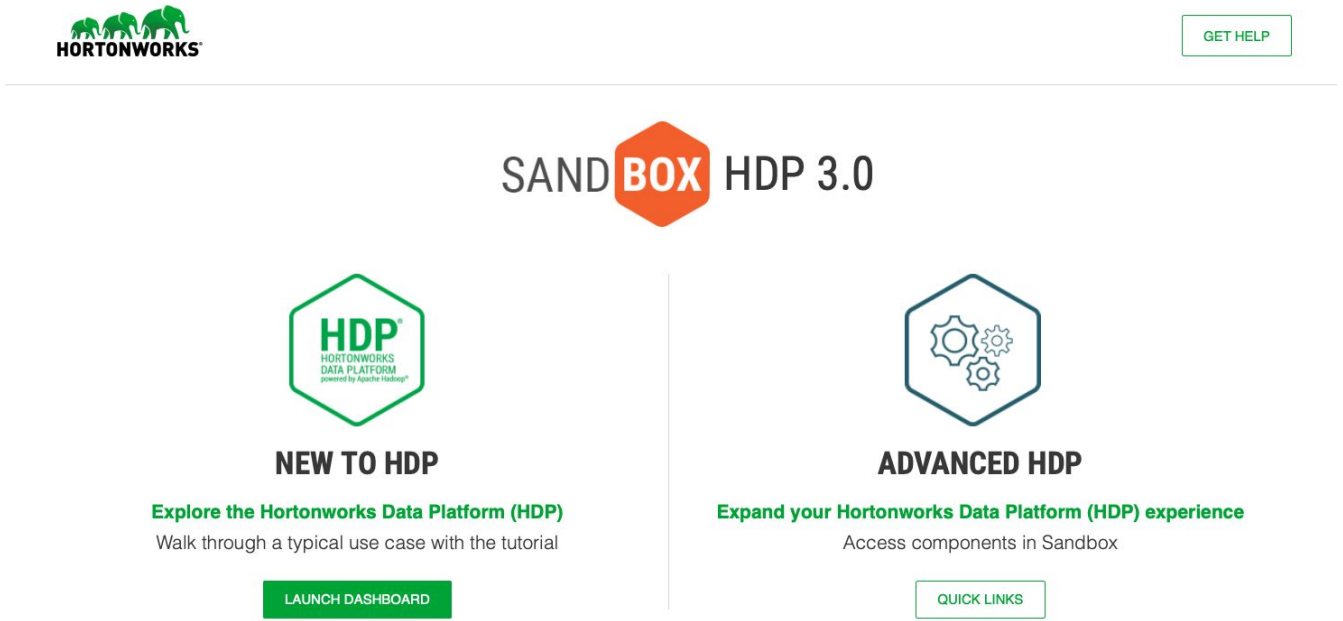
docker ps -> ищем id контейнера с hdp

docker exec -it a5c86792bcd1 /bin/bash -> подключаемся к контейнеру

ambari-admin-password-reset -> устанавливаем пароль

Начало работы

http://localhost:1080/



Считаем слова на Python

Цель - реализовать mapreduce на пальцах

Как - используем Python и Hadoop Streaming API для передачи данных через потоковый ввод-вывод

Считаем слова на Python

```
#!/usr/bin/env python
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

Считаем слова на Python

```
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
```

```
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
```

```
# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Копируем файлы

docker cp onegin.txt a5c86792bcd1:/tmp

docker cp mapper.py a5c86792bcd1:/tmp

docker cp reducer.py a5c86792bcd1:/tmp

Подключаемся к контейнеру

docker exec -it a5c86792bcd1 /bin/bash

Подключаемся к контейнеру

[root@sandbox-hdp /]# cd /tmp/

[root@sandbox-hdp tmp]# mkdir test

[root@sandbox-hdp tmp]# cp mapper.py test

[root@sandbox-hdp tmp]# cp onegin.txt test

[root@sandbox-hdp tmp]# cp reducer.py test

[root@sandbox-hdp tmp]# cd test/

[root@sandbox-hdp test]# ll

Работаем в контейнере

[root@sandbox-hdp test]# chmod a+x mapper.py

[root@sandbox-hdp test]# chmod a+x reducer.py

[root@sandbox-hdp test]# echo "foo foo quux labs foo bar quux" | ./mapper.py

foo 1

foo 1

quux 1

...

Работаем в контейнере

[root@sandbox-hdp test]# hdfs dfs -copyFromLocal . /tmp/test

[root@sandbox-hdp test]# hdfs dfs -ls /tmp/test/

Found 3 items

-rw-r--r--   1 root hdfs   552 2019-08-25 10:19 /tmp/test/mapper.py

-rw-r--r--   1 root hdfs   1029830 2019-08-25 10:19 /tmp/test/onegin.txt

-rw-r--r--   1 root hdfs   1044 2019-08-25 10:19 /tmp/test/reducer.py

Запускаем map-reduce

```
[root@sandbox-hdp hadoop]# /bin/hadoop jar
/usr/hdp/current/hadoop-mapreduce-client/hadoop-streami
ng.jar \

> -file /tmp/test/mapper.py -mapper /tmp/test/mapper.py \

> -file /tmp/test/reducer.py -reducer /tmp/test/reducer.py \

> -input /tmp/test/onegin.txt -output /tmp/test/output
```

Запускаем map-reduce

19/08/25 10:35:34 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.

...

19/08/25 10:35:40 INFO mapreduce.Job: The url to track the job: http://sandbox-hdp.hortonworks.com:8088/proxy/application_1566726469154_0002/

19/08/25 10:35:40 INFO mapreduce.Job: Running job: job_1566726469154_0002

Отслеживаем

http://localhost:8088/proxy/application_1566726469154_0002/

MapReduce Job job_1566726469154_0002

Job Overview	
Job Name:	streamjob8703753639274997889.jar
User Name:	root
Queue Name:	default
State:	RUNNING
Uberized:	false
Started:	Sun Aug 25 10:41:38 UTC 2019
Elapsed:	17sec

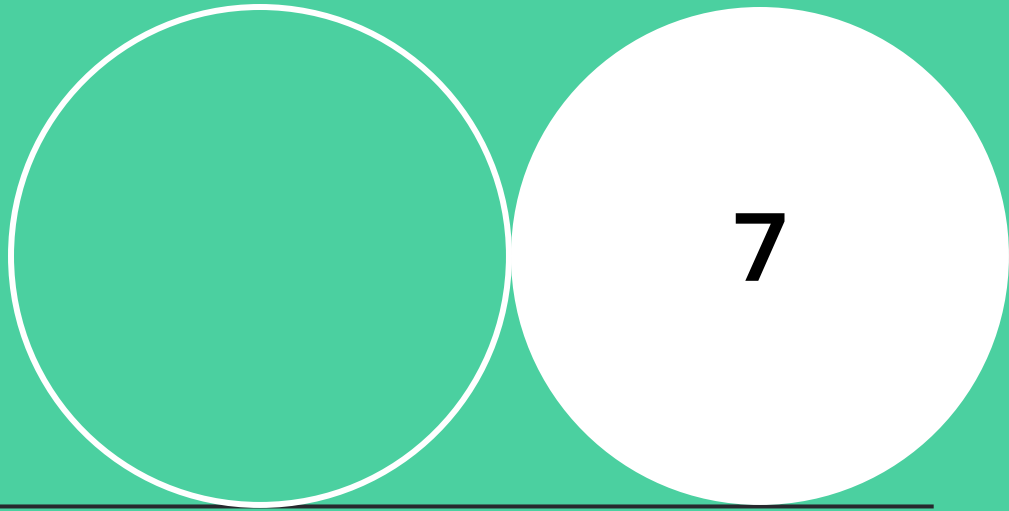
ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sun Aug 25 10:35:57 UTC 2019	sandbox-hdp.hortonworks.com:8042	logs

Task Type	Progress	Total	Pending	Running	Complete
Map		2	0	0	2
Reduce		1	0	1	0

Attempt Type	New	Running	Failed	Killed	Successful
Maps	0	0	0	0	2
Reduces	0	1	0	0	0

Итоги



Алексей Кузьмин

Технологии работы с большими данными

нетология

Что мы сегодня узнали

- Технологии больших данных нужны только если данных действительно много
- MapReduce – хороший подход для организации процессинга больших данных
- Hadoop – opensource проект, лидер на рынке больших данных
- Spark – расширяет и ускоряет Hadoop
- Установить Hadoop можно из одной из сборок

Домашнее задание

Алексей Кузьмин

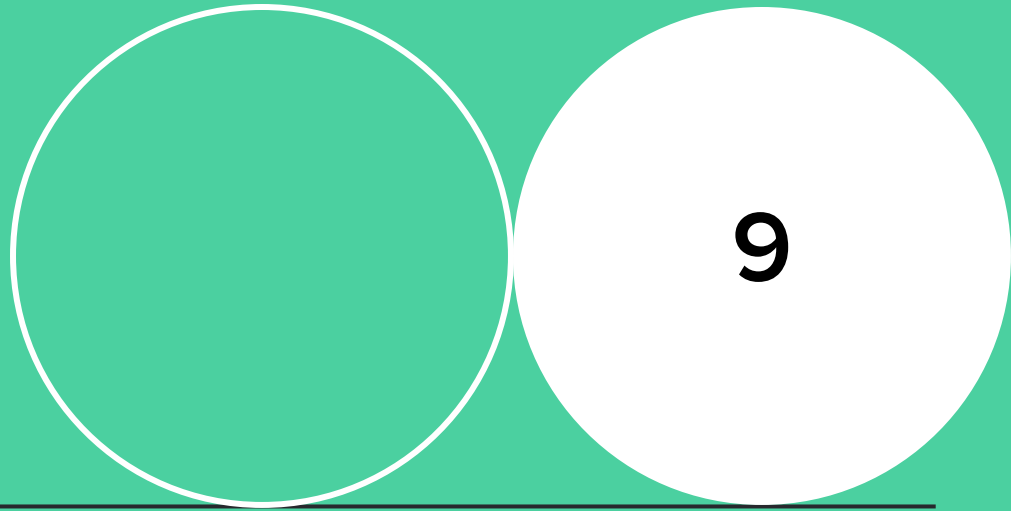
Технологии работы с большими данными

нетология

Домашнее задание

- Прочитать про операторы Spark. Прислать ответы на вопросы
- Какие команды отвечают за:
  - Сохранение результата в текстовый файл (Это Action или Transformation?)
  - Как получить первые n-элементов массива (Это Action или Transformation?)
  - Объединить два RDD в один (Это Action или Transformation?)
  - В чем разница между Reduce и CoGroup-операторами (Это Action или Transformation?)
- Нарисовать DAG для Spark’а для подсчета количества уникальных слов в файле

Полезные материалы



Алексей Кузьмин

Технологии работы с большими данными

нетология

Полезные материалы

- https://www.ozon.ru/context/detail/id/148770377/
- https://www.ozon.ru/context/detail/id/33061124/
- https://habr.com/ru/company/dca/blog/267361/

Спасибо за внимание

Алексей
Кузьмин

нетология