

Лекция 5

HADOOP & SPARK



Алексей Кузьмин

Директор разработки; Data Scientist

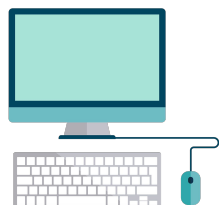
ДомКлик.ру



aleksej.kyzmin@gmail.com

Работа с данными

Источники
данных



Сбор данных
Лекция 7



SQL-БД
Лекция 1

Not Only SQL

NoSQL
Лекция 4



hadoop
Map Reduce
Лекция 5-6

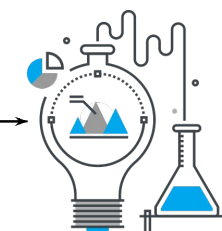
Мотивация Big Data
Лекция 3



Люди и процессы
Лекция 8



Примеры кейсов
Лекция 9



Data Science
Лекция 2



Отчеты
Лекция 1



Модели
Лекция 2

План

Сегодня - теоретическая часть

В следующий раз - практика Spark (PySpark)

Принципы работы с большими данными

- Независимое обрабатываем независимо
 - Это позволит обрабатывать независимые данные параллельно
- Принцип локальности данных
 - Обрабатываем там же, где храним
- Пошаговая работа
 - Сложный процесс обработки можно декомпозировать на несколько простых.

Hadoop



- Проект "The Apache™ Hadoop™" разрабатывает open-source ПО для отказоустойчивых, масштабируемых и распределенных вычислений
- – Работает с BigData на обычных серверах
- – Сильное open-source комьюнити
- – Много различных продуктов и средств используют Hadoop

Кластер Hadoop

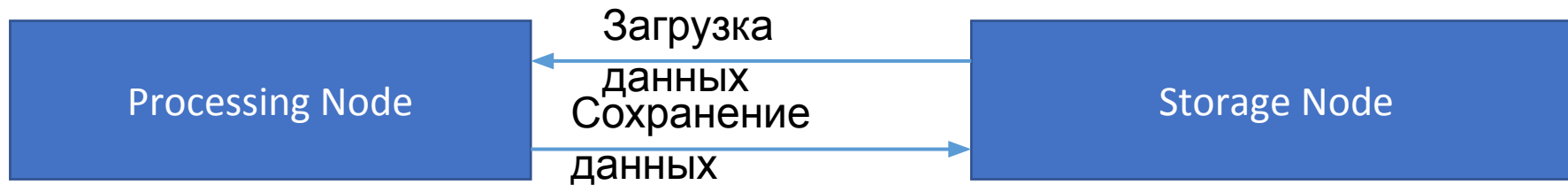
- “Дешевое” обычное железо (Commodity Hardware)
- Соединенное по сети
- Расположено в одном месте – Сервера в стойках в датацентре

Принципы Hadoop

- Горизонтальное масштабирование вместо вертикального
- Отправляем код к данным
- Уметь обрабатывать падения и отказы оборудования
- Инкапсуляция сложности работы распределенных и многопоточных приложений

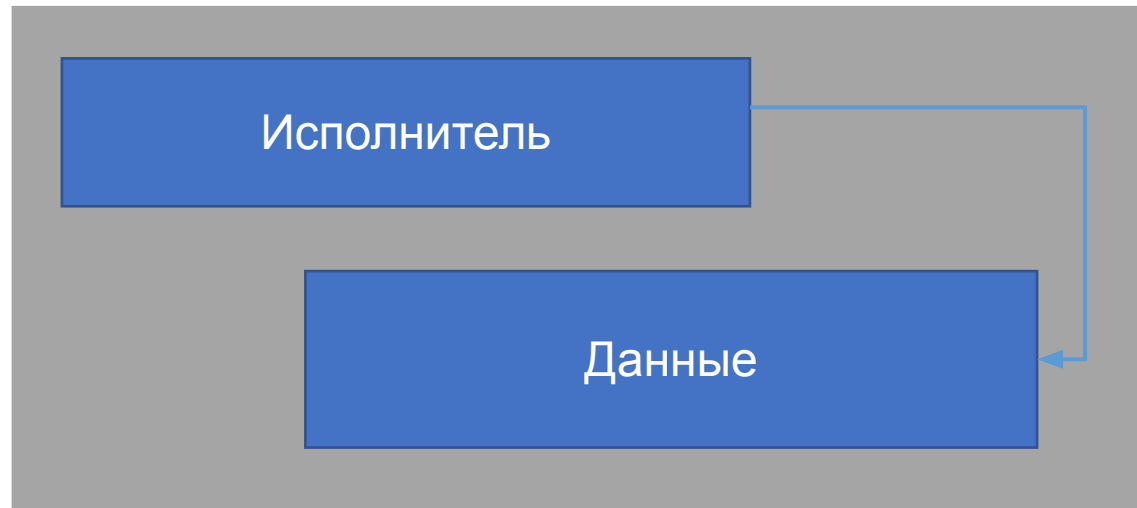
Доставка кода к данным

- Традиционная архитектура системы обработки данных
 - – Ноды системы разделяются на вычислительные и стораджи, соединяются высокоскоростным линком
 - – Многие приложения обработки данных являются CPU-bound, что приводит к проблемам с сетью



Доставка кода к данным

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



Узел
Hadoop

Отказы оборудования

- Чем больше количество машин, тем чаще будут отказы железа
 - – На больших кластерах отказы бывают несколько раз в день
- • Hadoop разрабатывался с учетом отказов железа
 - – Репликация данных
 - – Перезапуск тасков

Экосистема

- Изначально Hadoop был известен в основном из-за двух ключевых компонент:
 - – HDFS: Hadoop Distributed FileSystem
 - – MapReduce: Фреймворк распределенной обработки данных
- Сейчас, в дополнении к этому, также известны следующие продукты:
 - – Hbase: Column-oriented DB, поддержка последовательного и произвольного чтения, поддержка простых запросов
 - – Zookeeper: Highly-Available Coordination Service
 - – Oozie: Диспетчер задач для Hadoop
 - – Pig: Язык обработки данных и среда выполнения
 - – Hive: Data warehouse с SQL интерфейсом

—

HDFS

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.

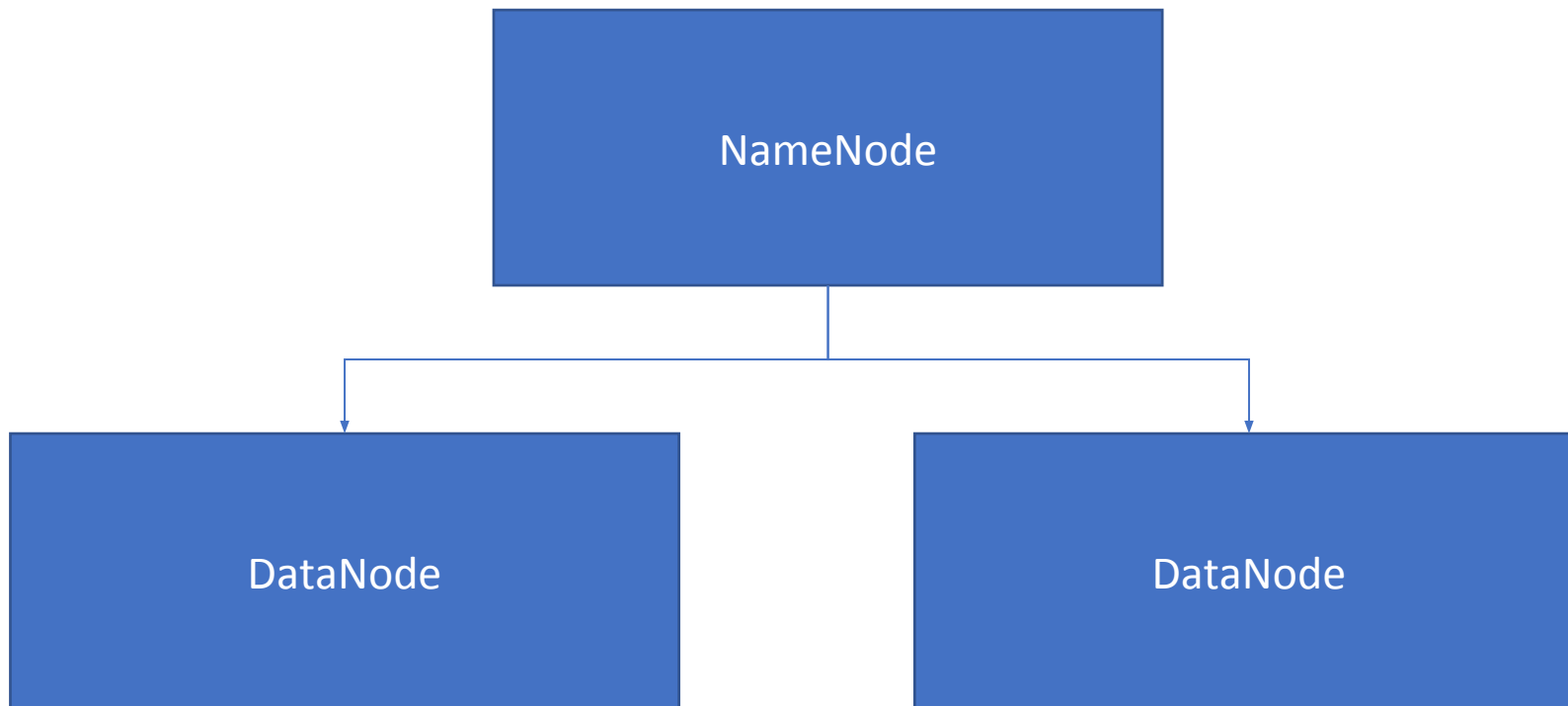
HDFS хорош для

- 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

Узлы HDFS

- Для управления файловой системой есть два основных типа узлов
 - – Namenode
 - Отвечает за файловое пространство (namespace), мета-информацию и расположение 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

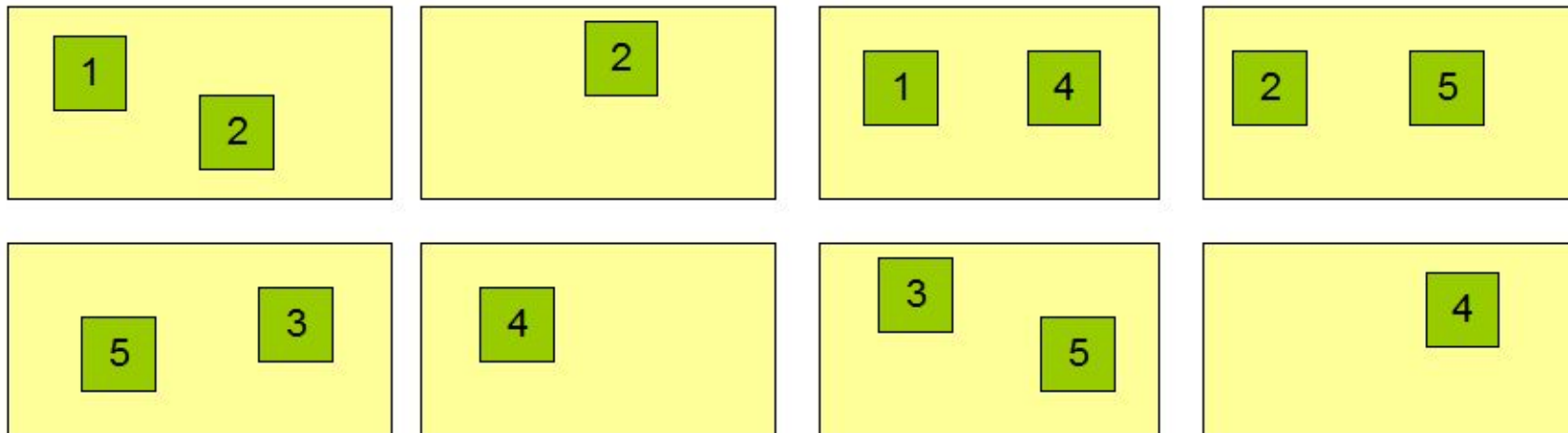
- Файлы в HDFS состоят из блоков
 - Блок – единица хранения данных
- Управляется через Namenode
- Хранится на Datanode
- Реплицируются по машинам в процессе записи
 - – Один и тот же блок хранится на нескольких Datanode
- Фактор репликации по умолчанию равен 3

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



Блоки в HDFS

- Стандартный размер блоков 64Мб или 128Мб
- Основной мотив этого – снизить стоимость *поиска* по сравнению со скоростью передачи данных

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

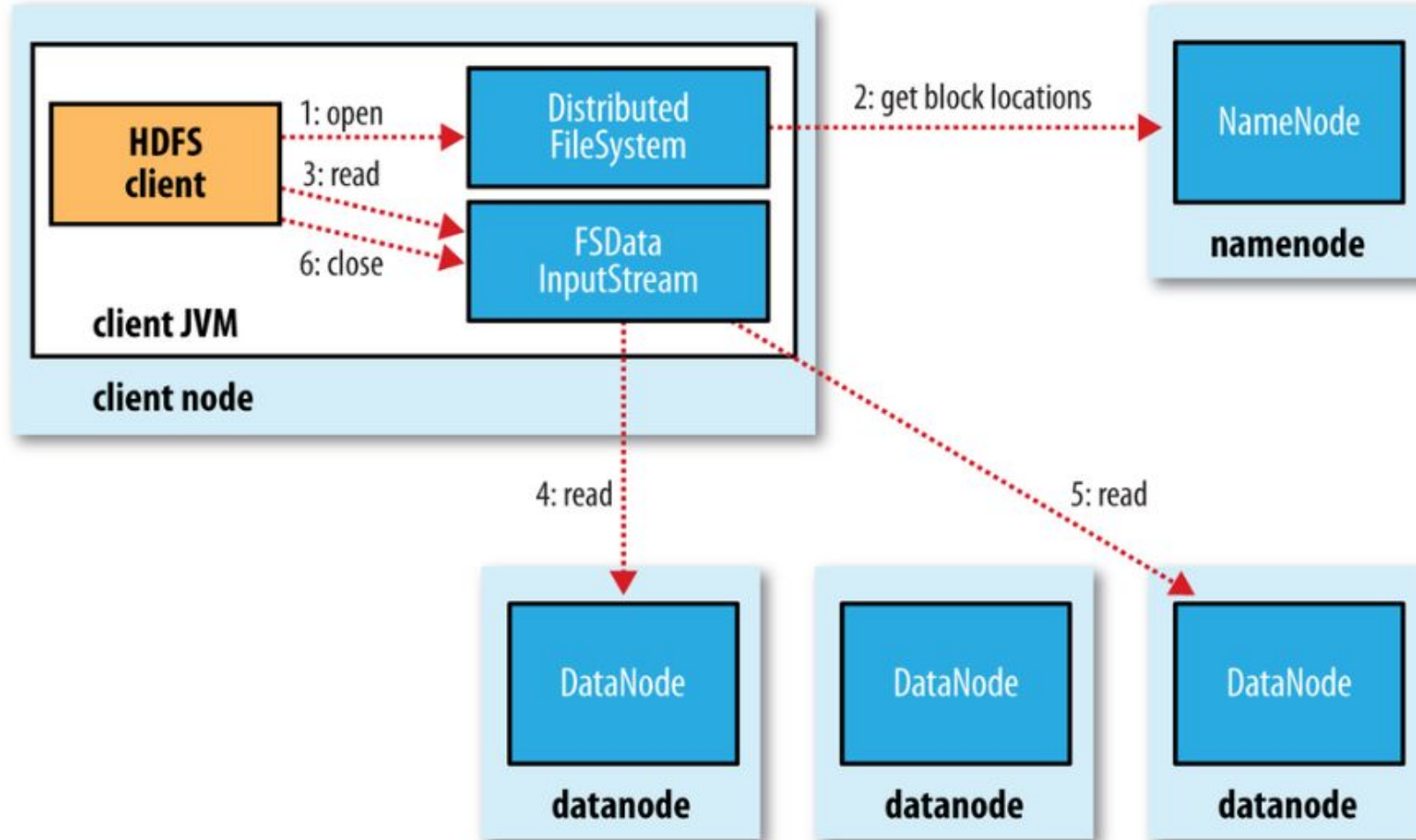
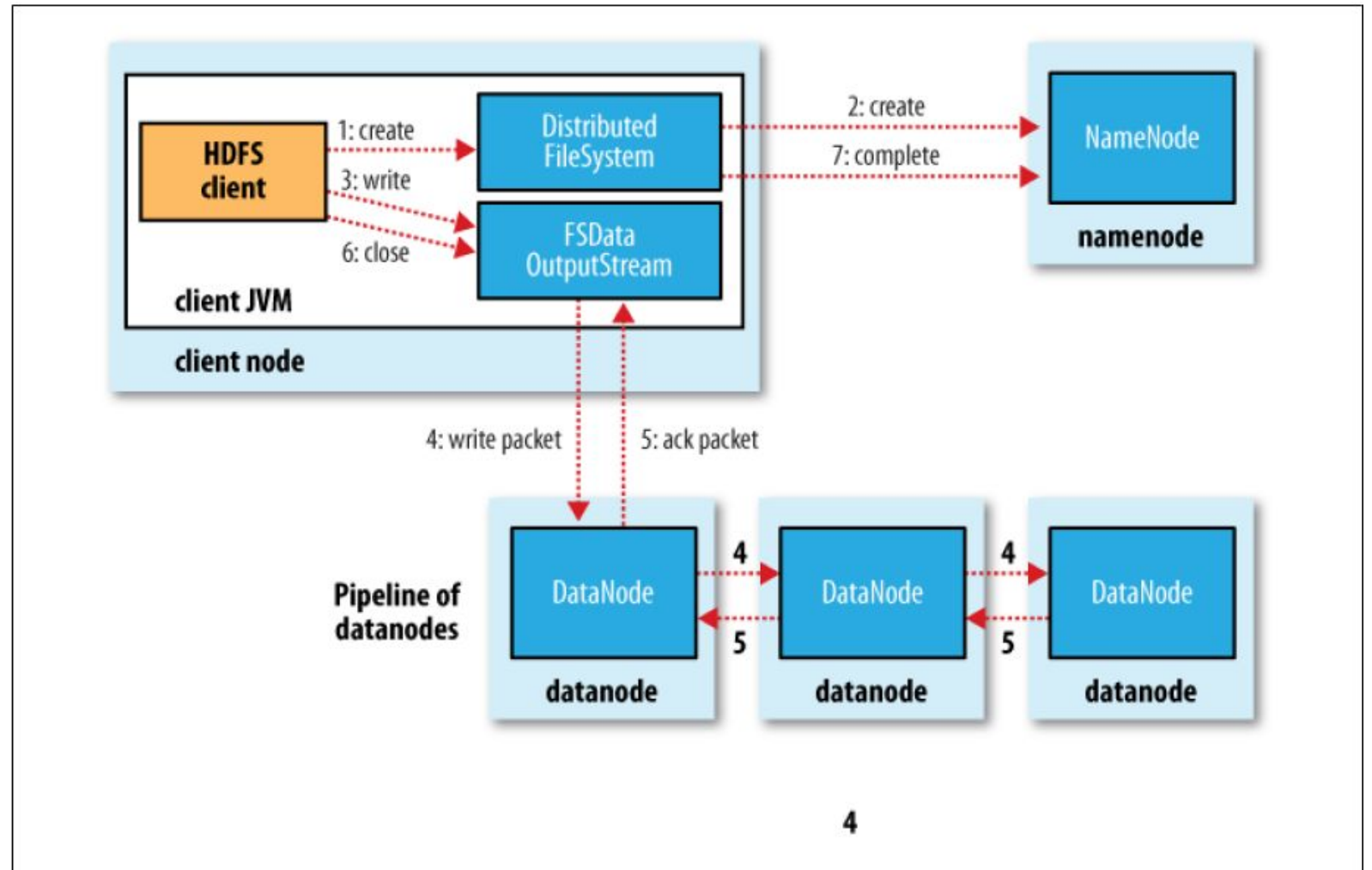


Figure 3-2. A client reading data from HDFS

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

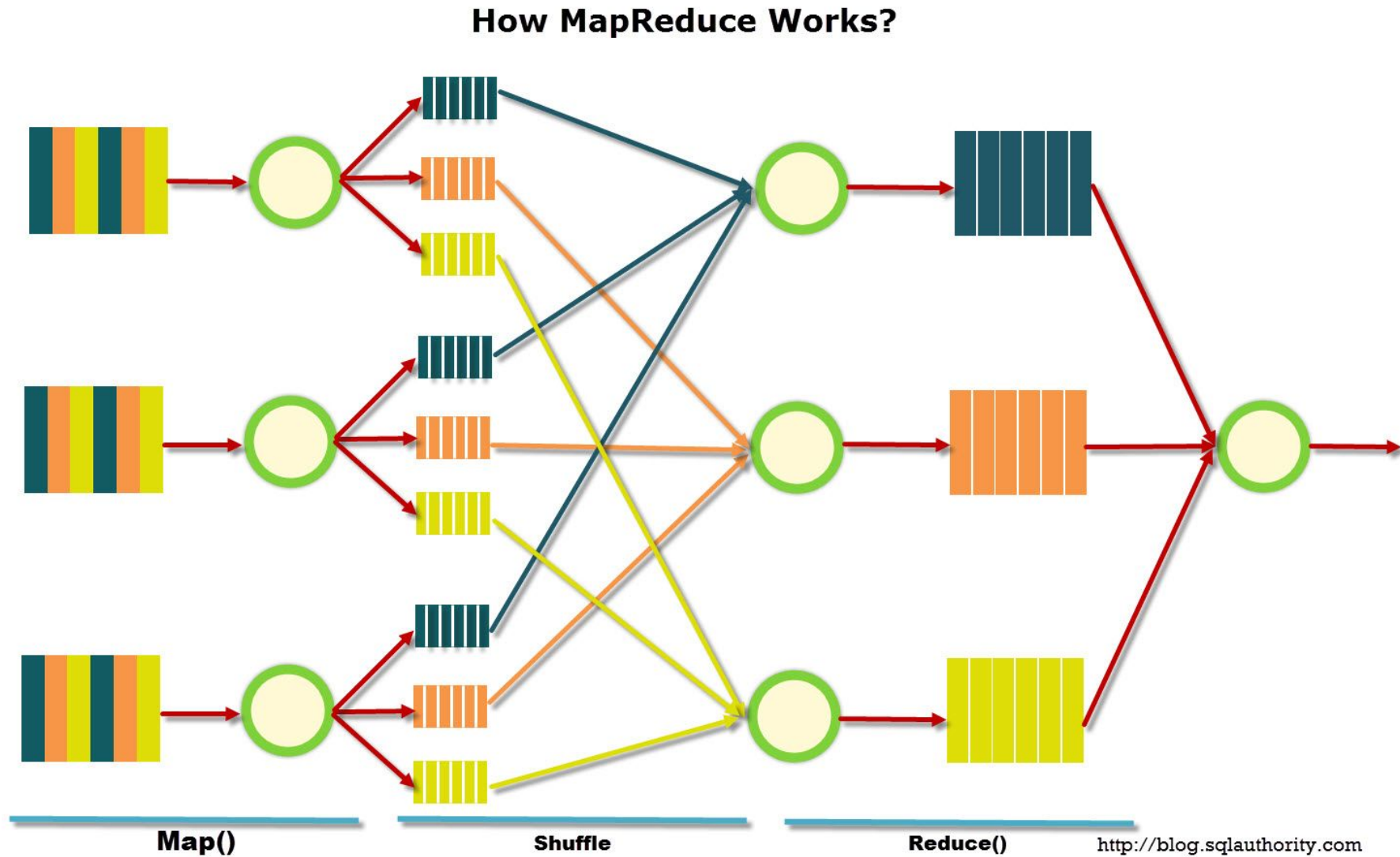
—

MapReduce

MapReduce

- Самая известная парадигма обработки больших данных
- Предложена компанией Google в 2004 году
- Имеет множество имплементаций (в том числе open source)

Схема работы



MapReduce

- **Map-шаг**

- Вход – исходный объект
- Выход – множество пар ключ-значение

- **Shuffle** – данные сортируются по ключу

- и распределяются по редьюсерам

- **Reduce-шаг**

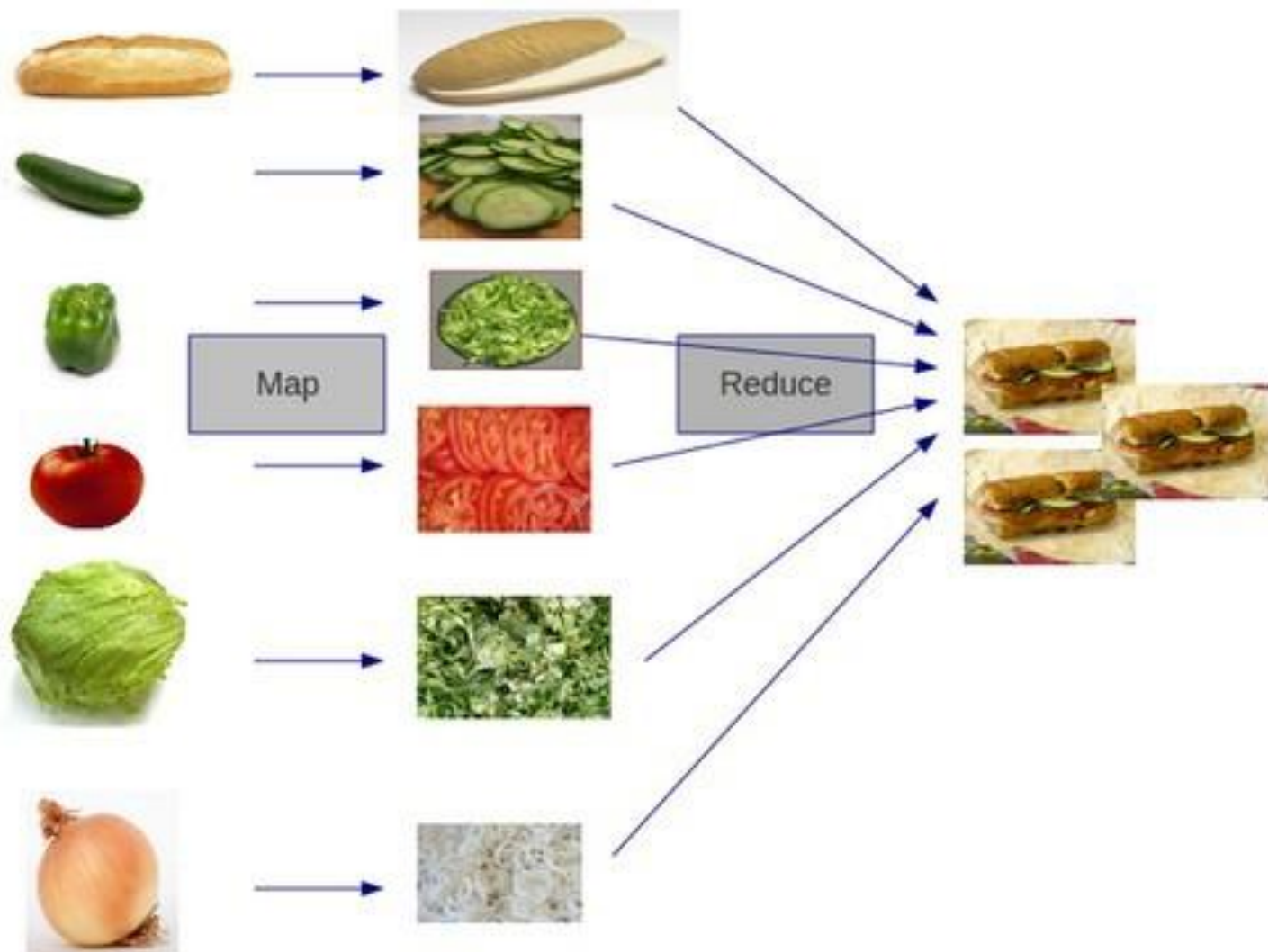
- Вход – Ключ -> список значений
- Выход – Ключ -> значение

Задача шага **map** – предобработать данные и выделить в них **ключ** - признак по которому данные будут агрегированы

Задача шага **shuffle** разложить данные по “корзинам” в соответствии с ключом

Данные обладающие одним ключом обрабатываются вместе.

Sandwich MapReduce



Map-шаг: порезать продукты на кусочки.

Shuffle – разобрать порезанные продукты по корзинкам (2 половинки хлеба, 3 кусочка огурца, 2 помидора и тд)

Reduce-по каждой из корзинок собрать готовый сэндвич.

Word Count

- Дано – файл со строками.
 - Одна строка = 1 документ
- Посчитать:
 - Сколько раз встречается каждое слово в исходном файле

Решение

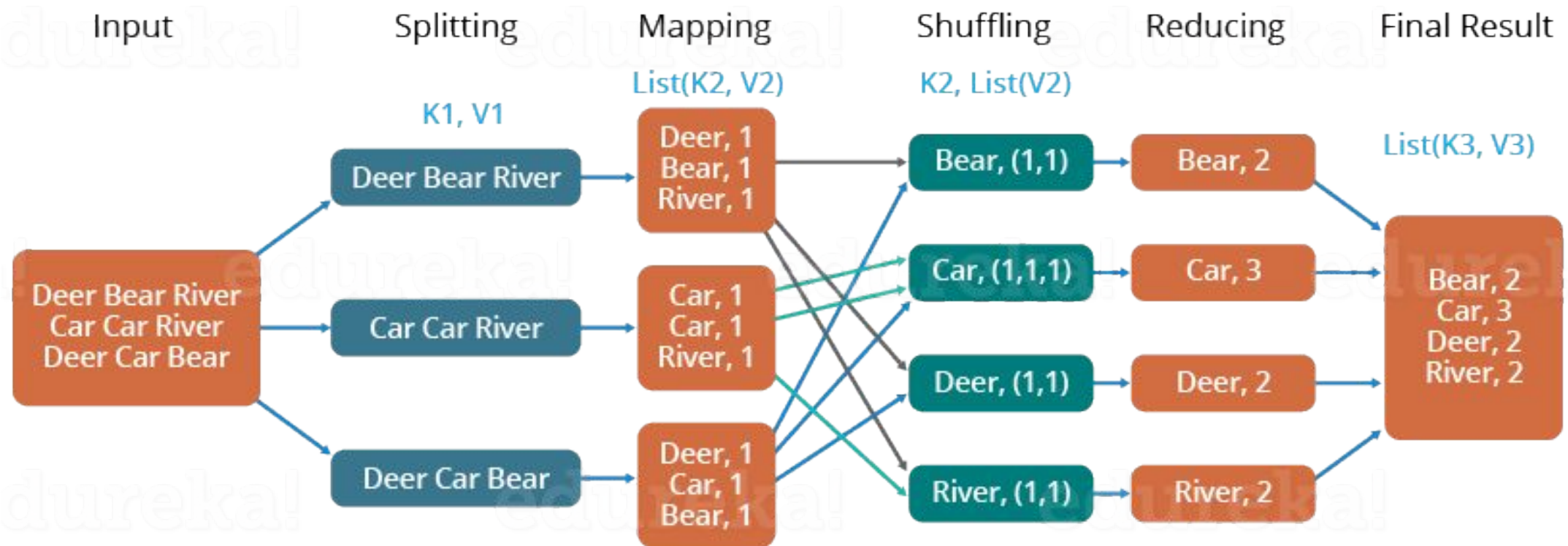
```
def map(string):  
    for word in string.split():  
        return (word, 1)  
  
def reduce(word, list_of_one):  
    return (word, sum(list_of_one))
```

Shuffle шаг реализуется фреймворком и не задается пользователем

WordCount

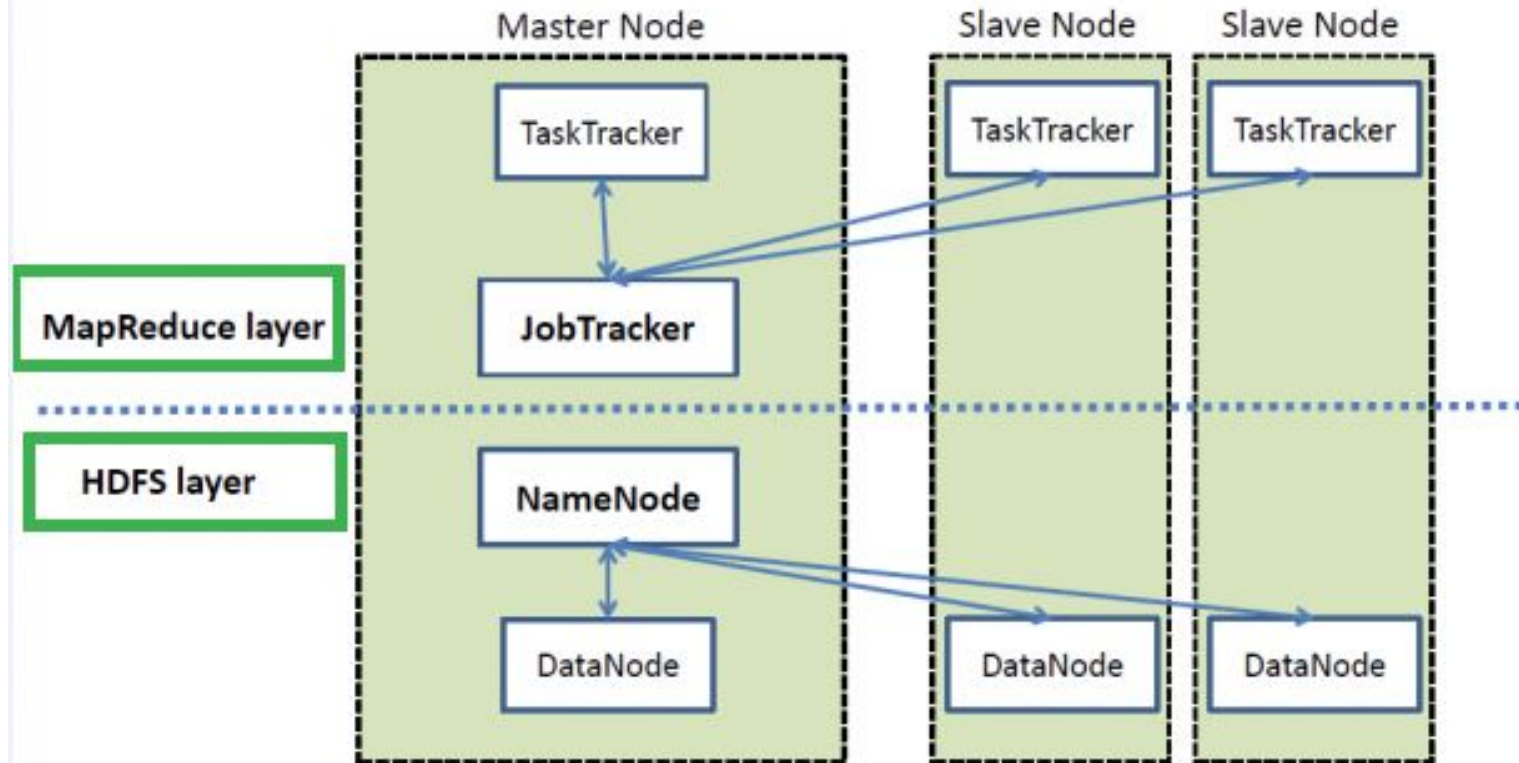
The Overall MapReduce Word Count Process

edureka!



MapReduce & HDFS

High Level Architecture of Hadoop



MapReduce “Runtime”

- Управление запуском задач
 - – Назначает воркерам задачи *map* или *reduce*
- Управление “*data distribution*”
 - – Перемещает код к данным
- – 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 task
- • Все работает поверх распределенной FS

MapReduce

- MapReduce работает поверх демонов
 - • *JobTracker* и *TaskTracker*
- • *JobTracker*
 - – Управляет запуском задач и определяет, на каком *TaskTracker* задача будет запущена
 - – Управляет процессом работы MapReduce задач (*jobs*)
 - – Мониторит прогресс выполнения задач
 - – Перезапускает зафейленные или медленные задачи
- • MapReduce имеет систему ресурсов основанную на слотах (*slots*)
 - – На каждом *TaskTracker* определяется, сколько будет запущено слотов
 - – Задача запускается в одном слоте
 - – $M \text{ мапперов} + R \text{ редьюсеров} = N \text{ слотов}$



Spark

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

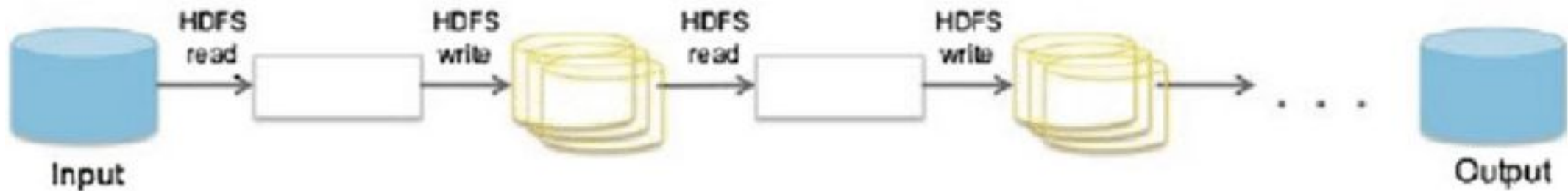
Мотивация

- Для решения обоих типов проблем требуется 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*

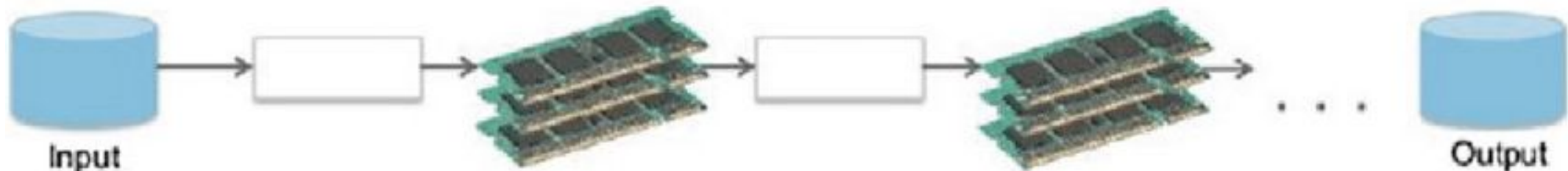
Решение

- Обработка и разделение данных в памяти (RAM)

Hadoop MapReduce: Data Sharing on Disk



Spark: Speed up processing by using Memory instead of Disks



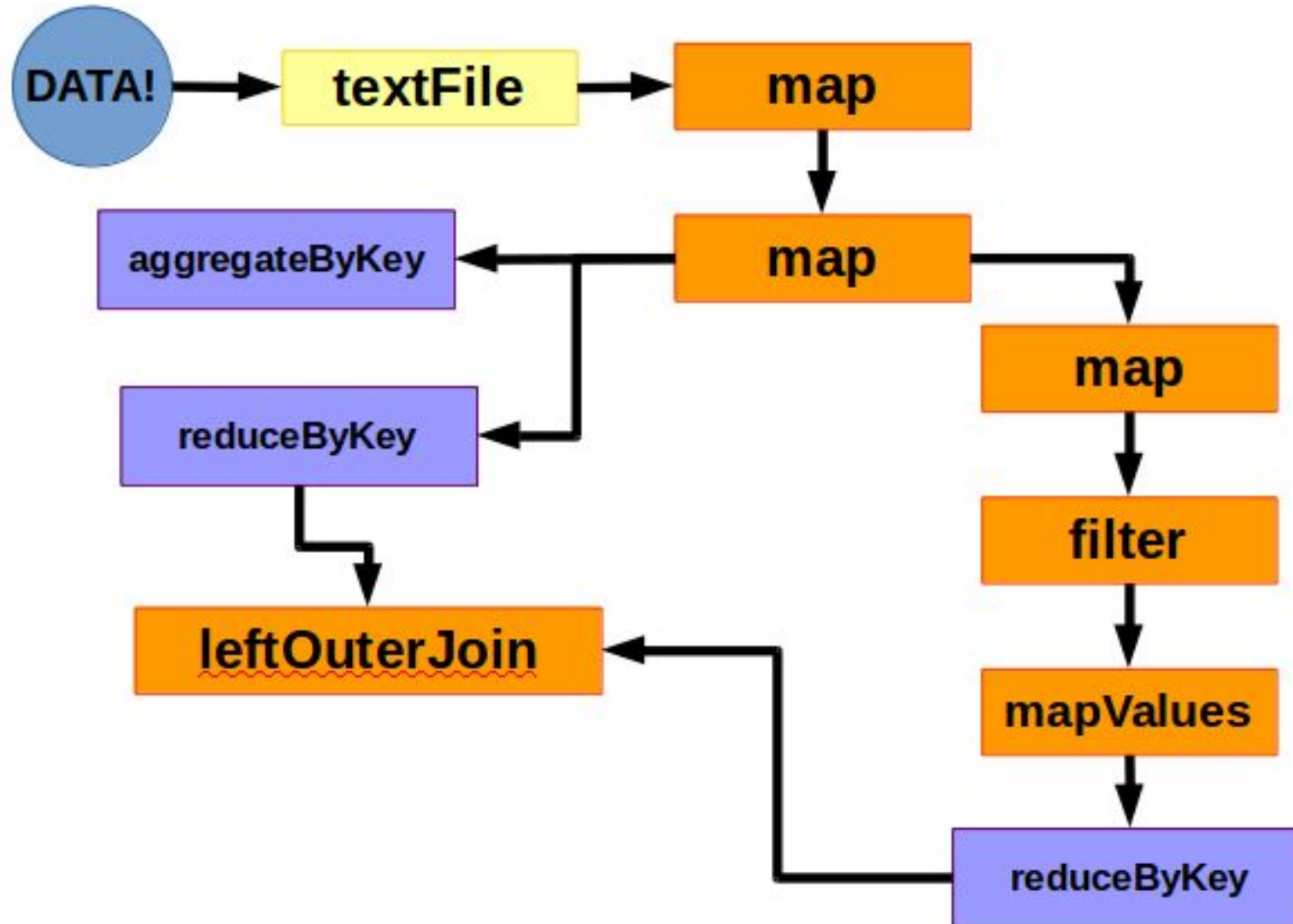
Resilient Distributed Datasets (RDD)

- 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 храниться на различных нодах кластера

Программная модель

- Основана на **parallelizable operators**
- Эти операторы являются функциями высокого порядка, которые выполняют *user-defined* функции параллельно
- Поток обработки данных состоит из любого числа **data sources, operators** и **data sinks** путем соединения их *inputs* и *outputs*

Программная модель



Основные операторы

- Существует два типа операторов – **transformations** и **actions**
- **Transformations:**
 - lazy-операторы, которые создают новые RDD
- **Actions:**
 - запускают вычисления и возвращают результат в программу или пишут данные во внешнее хранилище

Операторы

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

Контекст

- Основная точка входа для работы со Spark
- Доступна в *shell* как переменная *sc*
- В *standalone*-программах необходимо создавать отдельно

Shared Variable

- Когда Spark запускает выполнение функции параллельно как набор задач на различных нодах, то отправляется копия каждой переменной, используемой в функции, на каждый task
- Иногда нужно, чтобы переменная была общая между задачами или между задачами программой-драйвером

Shared Variable

- Обновления переменных не распространяются обратно в вызывающую
- Использование обычных *read-write* общих переменные между задачами неэффективно
 - – К примеру, необходимо отправить на каждую ноду большой датасет
- Есть два типа *shared variables*
 - – **broadcast variables**
 - – **accumulators**

BroadCast Variables

- Read-only переменные **кешируются** на каждой машине вместо того, чтобы отправлять копию на каждый task
- *Broadcast Variables* не отсылаются на ноду больше **одного раза**

Accumulators

- Могут быть только **добавлены**
- Могут использоваться для реализации **счетчиков** и **сумматоров**
- Таски, работающие на кластере, могут затем добавлять значение используя оператор **+=**

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

Другие инструменты Hadoop

Стек Hadoop

- **Hadoop-Common** - общие библиотеки hadoop
- **HDFS** – распределенная файловая система
- **Yarn** - Фреймворк управления распределенными задачами
- **Hadoop MapReduce** – Yarn-based реализация mapreduce
- Множество других hadoop-related проектов
 - Hbase, hive, pig, zookeeper, spark, cassandra (тысячи их 😊)

Apache Hive

- Транслятор SQL-like запросов в цепочки MapReduce запросов
- Очень сильно упрощает жизнь при разработке big data проектов
- Часто используется в enterprise решениях

Hbase

- Колоночная база данных
- 3х-мерная, ключ-значение, timestamp
- Позволяет объединить достоинства массивной обработки данных и записи данных по ключу.

Сборки Hadoop

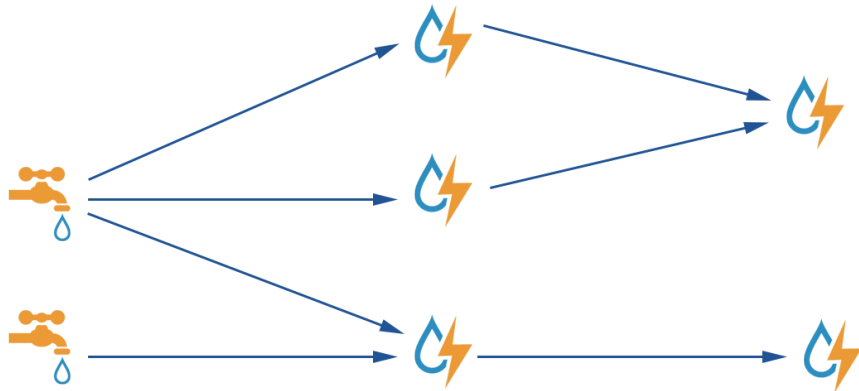
Несколько основных поставщиков Hadoop:

- [Cloudera](#)
- [Hortonworks](#)
- [MapR](#)

Потоковая обработка данных

- MapReduce – оффлайн алгоритм пакетной обработки данных
- Не подходит для задач где данные поступают быстро и на них необходимо быстро реагировать
- Примеры
 - Автомобильные пробки
 - Поиск мошеннических операций в банке

Потоковая обработка данных



Мини практика (для заинтересованных)

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/

[GET HELP](#)

SAND  HDP 3.0



NEW TO HDP

Explore the Hortonworks Data Platform (HDP)

Walk through a typical use case with the tutorial

[LAUNCH DASHBOARD](#)

ADVANCED HDP

Expand your Hortonworks Data Platform (HDP) experience

Access components in Sandbox

[QUICK LINKS](#)

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:
```

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

```
# 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
```


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-streaming.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 | <div><div></div></div> | | 2 | 0 | | 0 | | 2 | | |
| Reduce | <div><div></div></div> | | 1 | 0 | | 1 | | 0 | | |
| Attempt Type | New | | Running | | Failed | | Killed | | Successful | |
| Maps | 0 | 0 | 0 | | 0 | | 2 | | | |
| Reduces | 0 | 1 | 0 | | 0 | | 0 | | | |

Полезные материалы

- Hadoop: The Definitive Guide, Tom White
- Hadoop in Action, Chuk Lam

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
- Не для любой задачи подходит MapReduce

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



НЕТОЛОГИЯ
групп

Спасибо за
внимание!

Алексей Кузьмин



aleksej.kyzmin@gmail.com