Министерство образования и науки Российской Федерации

Калужский филиал

федерального государственного бюджетного образовательного учреждения высшего образования

# «Московский государственный технический университет имени Н.Э. Баумана

**(национальный исследовательский университет)»**

(КФ МГТУ им. Н.Э. Баумана)

**Ю.С. Белов, С.С. Гришунов**

ОСНОВЫ SPARK. УСТАНОВКА SPARK. ОСНОВНЫЕ КОМАНДЫ ДЛЯ РАБОТЫ С RDD

Методические указания к выполнению домашней работы по курсу «Технологии обработки больших данных»

Калуга - 2024

# ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_bookmark0)

[ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ](#_bookmark1) [ВЫПОЛНЕНИЯ 5](#_bookmark1)

[КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ,](#_bookmark2) [ИССЛЕДОВАНИЯ 6](#_bookmark2)

[УСТАНОВКА SPARK 7](#_bookmark4)

[РАБОТА С ДАННЫМИ В SPARK 10](#_bookmark6)

[ЗАДАНИЕ НА ДОМАШНЮЮ РАБОТУ 15](#_bookmark9)

[ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ 15](#_bookmark10)

[ВАРИАНТЫ ЗАДАНИЙ 15](#_bookmark11)

[КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ 21](#_bookmark12)

[ФОРМА ОТЧЕТА ПО ДОМАШНЕЙ РАБОТЕ 21](#_bookmark13)

[ОСНОВНАЯ ЛИТЕРАТУРА 22](#_bookmark14)

[ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА 22](#_bookmark15)

# ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения курса «Технологии обработки больших данных» на кафедре «Программное обеспечение ЭВМ, информационные технологии и прикладная математика» факультета фундаментальных наук Калужского филиала МГТУ им. Н.Э. Баумана. Методические указания, ориентированные на студентов 4-го курса направления подготовки 09.03.04 «Программная инженерия», описание установки и настройки Apache Spark, синтаксис основных команд, примеры скриптов и задание на выполнение домашней

работы.

Методические указания составлены для ознакомления студентов с платформой Apache Spark. Для выполнения домашней работы студенту необходимы минимальные знания по программированию на высокоуровневом языке программирования, имеющего интеграцию с Apache Spark (Java, Python, Scala и др.).

# ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения домашней работы является формирование практических навыков работы с платформой Apache Spark для обработки больших данных.

Основными задачами выполнения домашней работы являются:

1. Изучить основы Apache Spark.
2. Научиться устанавливать и конфигурировать Spark.
3. Уметь работать с RDD.
4. Получить навыки написания программ для обработки больших данных.

[Результатами](#_bookmark9) работы являются:

* Входные файлы с данными
* Программа, реализующая рекомендательную систему
* Выходные файлы с результатами вычислений
* Подготовленный отчёт

# КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

Apache Spark — фреймворк с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop. В отличие от классического обработчика из ядра Hadoop, реализующего двухуровневую концепцию MapReduce с хранением промежуточных данных на накопителях, Spark работает в парадигме резидентных вычислений — обрабатывает данные в оперативной памяти, благодаря чему позволяет получать значительный выигрыш в скорости работы для некоторых классов задач[7], в частности, возможность многократного доступа к загруженным в память пользовательским данным делает библиотеку привлекательной для алгоритмов машинного обучения.

Spark в настоящее время предоставляет API для Java, Scala, Python, R. Изначально написан на Scala, впоследствии добавлена существенная часть кода на Java для предоставления возможности написания программ непосредственно на Java.

Состоит из ядра и нескольких расширений, таких как:

1. Spark SQL (позволяет выполнять SQL-запросы над данными)
2. Spark Streaming (надстройка для обработки потоковых данных)
3. Spark MLlib (набор библиотек машинного обучения)
4. GraphX (предназначено для распределённой обработки графов).

Может работать как в среде кластера Hadoop под управлением YARN, так и без компонентов ядра Hadoop, поддерживает несколько распределённых систем хранения — HDFS, OpenStack Swift, NoSQL-СУБД Cassandra, Amazon S3.

# УСТАНОВКА SPARK

**Необходимое ПО**

Для установки Spark необходима операционная система Linux. Далее будет рассмотрена установка и настройка [Spark 3.5.0](https://spark.apache.org/downloads.html) под ОС Ubuntu 22.04.3 LTS.

Для работы потребуется следующее программное обеспечение: Java 8+, Python 3.10+

# Создание учётной записи для Spark

Для запуска Spark будет использоваться отдельная учётная запись Linux. Этот шаг не является обязательным. Создадим группу spark и добавим в неё пользователя sparkusr, также предоставим новому пользователю права sudo, для этого необходимо выполнить команды:

$ sudo addgroup spark

$ sudo adduser --ingroup spark sparkusr

$ sudo usermod -aG sudo sparkusr

Предполагается, что все дальнейшие действия будут выполняться от только что созданного пользователя.

# Настройка SSH

Сгенерируем новый ssh ключ:

$ ssh-keygen -t rsa -P ""

Добавим созданный ключ в список авторизованных:

$ cat $HOME/.ssh/id\_rsa.pub >> $HOME/.ssh/authorized\_keys

Для проверки подключимся к localhost:

$ hduser@ubuntu:~$ ssh localhost

The authenticity of host 'localhost (::1)' can't be established.

RSA key fingerprint is d7:87:25:47:ae:02:00:eb:1d:75:4f:bb:44:f9:36:26. Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'localhost' (RSA) to the list of known hosts

# Распаковка

Скачать файлы Spark можно с официального сайта, либо выполнив следующую команду:

$ sudo wget <https://dlcdn.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz>

Далее нужно распаковать архив и переместить фалы в каталог /usr/local/spark:

$ sudo mv spark-3.5.0-bin-hadoop3.tgz /usr/local/

$ cd /usr/local/

$ sudo tar xzf spark-3.5.0-bin-hadoop3.tgz

$ sudo mv spark-3.5.0-bin-hadoop3 spark

Также необходимо дать пользователю sparkusr права создателя на директорию:

$ sudo chown -R sparkusr:spark spark

# Настройка переменных окружения

В файл $HOME/.bashrc нужно добавить следующие переменные:

export JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64/jre/

export SPARK\_HOME=/usr/local/spark

export PATH=$SPARK\_HOME/bin:$PATH

После этого можно запустить spark командой:

$ pyspark

По выполнению этой команды в терминале откроется оболочка для ввода spark-команд (рис. 1).

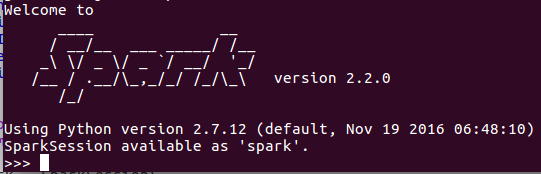


Рис. 1. Окно ввода spark-команд

Для выхода из режима ввода команд необходимо ввести команду exit()

# РАБОТА С ДАННЫМИ В SPARK

**RDD**

В Spark вводится концепция RDD (*Resilient Distributed Dataset*, устойчивый распределенный набор данных) – неизменяемая отказоустойчивая распределенная коллекция объектов, которые можно обрабатывать параллельно. В RDD могут содержаться объекты любых типов; RDD создается путем загрузки внешнего набора данных или распределения коллекции из основной программы.

В RDD поддерживаются операции двух типов (рис. 2):

1. Трансформации – это операции (например, отображение, фильтрация, объединение и т.д.), совершаемые над RDD; результатом трансформации становится новый RDD, содержащий ее результат.
2. Действия – это операции (например, редукция, подсчет и т.д.), возвращающие значение, получаемое в результате некоторых вычислений в RDD.

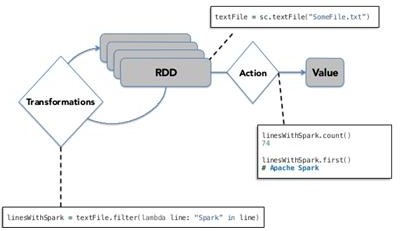


Рис. 2. Операции над RDD

# Трансформации

Результатом применения данной операции к RDD является новый RDD. Как правило, это операции, которые каким-либо образом преобразовывают элементы данного dataset. Вот неполный список самых распространенных преобразований, каждое из которых возвращает новый RDD (полный список представлен в документации Apache Spark):

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

# Действия

Действия применяются тогда, когда необходимо материализовать результат — как правило, сохранить данные на диск, либо вывести часть данных в консоль. Вот список самых распространенных действий, которые можно применять над RDD:

* **.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 RDD в виде массива. Как правило, это применяется в случаях, когда данных в RDD уже мало (применены различные фильтры и преобразования) — и необходима визуализация, либо дополнительный анализ данных.
* **.take(n)** — возвращает в виде массива первые n элементов RDD
* **.count()** — возвращает количество элементов в RDD
* **.reduce(function)** — знакомая операция для тех, кто знаком с **MapReduce**. Из механизма этой операции следует, что функция function (которая принимает на вход 2 аргумента возвращает одно значение) должна быть обязательно коммутативной и ассоциативной.

# Spark Shell

[Spark Shell](#_bookmark5) позволяют обрабатывать данные, находящиеся на разных узлах кластера, и Spark автоматически распределяет эту обработку.

При запуске Spark необходимо создать **SparkContext** (это объект, который отвечает за реализацию более низкоуровневых операций с кластером), который при запуске **Spark Shell** создается автоматически и доступен сразу (обьект **sc**)

Для Python используется оболочка $SPARK\_INSTALL/bin/pyspark, для Scala $SPARK\_INSTALL/bin/spark-shell

# Загрузка данных

Загружать данные в Spark можно двумя путями:

1. Непосредственно из локальной программы с помощью функции

**.parallelize(data):**

>>> data = [1,2,3,4,5]

>>> myRDD = sc.parallelize(data)

1. Из поддерживаемых хранилищ (например, HDFS) с помощью функции **.textFile(path)**:

>>> myRDD = sc.textFile(“hdfs://…”)

# Пример

Рассмотрим стандартный пример для обработки больших данных – приложение для подсчета количества слов. Создадим файл example.py:

import sys

from pyspark.sql import SparkSession spark = SparkSession\

.builder\

.appName("WordCount")\

.getOrCreate()

text\_file = spark.sparkContext.textFile("../file.txt") counts = text\_file.flatMap(lambda line: line.split(",")) \

.map(lambda word: (word, 1)) \

.reduceByKey(lambda a, b: a + b) counts.saveAsTextFile("./result") spark.stop()

Данный код не только легче и проще писать и читать, но также он выполняется на кластере гораздо быстрее, чем стандартный Hadoop MapReduce, т.к. Hadoop MapReduce записывает промежуточные результаты каждой операции на диск, а следующая операция по новой их считывает – работа с дисками является самым узким местом Hadoop MapReduce – в то время как Apache Spark, минимизирует количество операций чтения/записи на диск и сохраняет промежуточные результаты в оперативной памяти серверов.

Рассмотрим более подробно использованные команды:

* text\_file = spark.sparkContext.textFile("../file.txt") – [загрузка](#_bookmark8) данных из файла file.txt
* text\_file.flatMap(lambda line: line.split(",")) – разбиение строки на слова
* .map(lambda word: (word, 1)) – создаем пары ключ-значение, каждому ключу присваиваем значение 1
* .reduceByKey(lambda a, b: a + b) – объединяем пары с одинаковыми ключами, складывая значения
* counts.saveAsTextFile("./result") – результат [сохраняем](#_bookmark7) в файл директории result

Для запуска необходимо команду spark-submit с указанием скрипта и файла с входными данными:

$ SPARK\_INSTALL/bin/spark-submit example.py file.txt

Результаты выполнения скрипта будут сохранены в файл bin/result/part-00000.

При содержимом файла file.txt:

*sun, fun, cat, dog, sun, cat*

*fun, fun, fun, fun, fun, fun*

Будет получен следующий результат:

*(u’fun’, 7)*

*(u’sun’, 2)*

*(u’dog’, 1)*

*(u’cat’, 2)*

# ЗАДАНИЕ НА ДОМАШНЮЮ РАБОТУ

Написать скрипт для платформы Apache Spark для решения задачи, указанной в варианте. В качестве входных текстовых файлов можно использовать книги в txt формате из библиотеки Project Gutenberg: www.gutenberg.org

Список стоп-слов: <https://github.com/Alir3z4/stop-words/blob/master/english.txt>

# ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ

Приложение может быть реализовано на любом языке высокого уровня, имеющем интеграцию с платформой Apache Spark (Java, Python, Scala и др.). В качестве источника данных можно пользоваться как файлами, размещёнными в HDFS, так и файлами в локальной файловой системе.

# ВАРИАНТЫ ЗАДАНИЙ

1. Подсчитать средний рейтинг фильма. Входный файл рейтингов имеет формат:

userId, movieId,rating,timestamp

Выполнить операцию объединения с файлом, содержащим названия фильмов. Данный файл имеет формат:

movieId, title, genres

Если для фильма указано несколько жанров, оставить только первый. Сгруппировать записи по жанру и подсчитать средний рейтинг жанра. Результат сохранить в файл:

genre, av\_rating Исходные файлы:

rating.csv

[movies.csv](https://github.com/clemsonciti/Introduction-to-Hadoop-data/blob/master/ml-latest-small/movies.csv)

1. Подсчитать вероятность взаимного появления двух слов в предложении. Если предложение состоит из слов “A B C”, то взаимно появляющимися парами будут:

(A, B)

(A, C)

(B, A)

(B, C)

(C, A)

(C, B)

Для оценки вероятности взаимного появления использовать метрику PMI (pointwise mutual information):

где – частота взаимного появления двух слов, - частота появления слова в отдельности.

Для реализации использовать подход «пар» (pairs approach). То есть, результат будет представлен в виде пары ключ-значение, где ключом будет пара слов (a,b), а значением – *PMI*(a,b).

Из результата должны быть удалены стоп-слова

1. Подсчитать вероятность взаимного появления двух слов в предложении. Если предложение состоит из слов “A B C”, то взаимно появляющимися парами будут:

(A, B)

(A, C)

(B, A)

(B, C)

(C, A)

(C, B)

Для оценки вероятности взаимного появления использовать метрику PMI (pointwise mutual information):

где – частота взаимного появления двух слов, - частота появления слова в отдельности.

Для реализации использовать подход «полосы» (stripe approach). То есть, результат будет представлен в виде пары ключ-значение, где ключом будет слово a, а значением – список-хэш, содержаший все слова с котороми есть взаимное появление слова a в файле (ключ – слово b, а значение – *PMI*(a,b)).

Из результата должны быть удалены стоп-слова.

1. Подсчитать средний рейтинг фильма. Входный файл рейтингов имеет формат:

userId, movieId,rating,timestamp

Выполнить операцию объединения с файлом, содержащим названия фильмов. Данный файл имее формат:

movieId, title, genres

Результат должен быть сохранен в файле в формате: movieId, title, av\_rating

Объединение выполнять согласно подходу Job-side join. В результате должны быть представлены 20 фильмов с самым высоким средним рейтингом

Исходныефайлы: [rating.csv](https://github.com/clemsonciti/Introduction-to-Hadoop-data/blob/master/ml-latest-small/ratings.csv) [movies.csv](https://github.com/clemsonciti/Introduction-to-Hadoop-data/blob/master/ml-latest-small/movies.csv)

1. Для заданного списка слов (поисковый запрос) среди множества из N документов найти 3 самых похожих. Для меры схожести использовать метрику TF·IDF. Результат не должен учитывать стоп-слова.

Метрика TF·IDF для слова s в документе d вычисляется:

*TF*  *IDF*  1  ln *TF*  \* ln *N*

*IDF*

где N – общее количество документов, TF – количество встречания слова s в документе d, IDF – количество документов, в которых встречается слово s.

Возможные этапы реализации:

Стадия 1:

Подсчет всех слов в каждом документе (по аналогии с WordCount).

Стадия 2:

Подсчет tf-idf каждого слова для каждого документа. Входными данными являются результаты стадии 1.

Стадия 3:

Подсчет нормализованных значений TF·IDF. Входными данными являются результаты стадии 2.

Стадия 4:

Подсчет схожести документов с поисковым запросом. Входными данными являются результаты стадии 3 и файл с запросом. Для каждого документа сложить все значения TF·IDF для слов, присутсвующих в запросе.

Стадия 5:

Отбор 3 самых подходящих документов.

1. Для каждого пользователя подсчитать среднюю оценку, которую он поставил всем фильмам. Входный файл рейтингов имеет формат:

userId, movieId,rating,timestamp

Не учитывать пользователей, которые поставили менее 20 оценок.

Результат отсортировать по убыванию рейтинга фильма и сохранить в файле в формате:

userId, av\_rating

Исходные файлы: [rating.csv](https://github.com/clemsonciti/Introduction-to-Hadoop-data/blob/master/ml-latest-small/ratings.csv)

1. Для двух текстовых файлов подсчитать количество слов, которые встречаются одновременно и в первом, и во втором файле. Результат сохранить в файл в виде пар ключ-значение, где ключ – количество общих слов, значение – само слово.
2. Подсчитать средний рейтинг фильма. Входный файл рейтингов имеет формат:

userId, movieId,rating,timestamp

Выполнить операцию объединения с файлом, содержащим названия фильмов. Данный файл имеет формат:

movieId, title, genres

Если для фильма указано несколько жанров, оставить только первый. Сгруппировать записи по жанру и подсчитать средний рейтинг жанра. Результат сохранить в файл:

genre, av\_rating Исходные файлы:

rating.csv

[movies.csv](https://github.com/clemsonciti/Introduction-to-Hadoop-data/blob/master/ml-latest-small/movies.csv)

1. Реализовать алгоритм кластеризации K-means.

Стадия map связывает каждую точку с ближайшим центром. Стадия Reduce выполняет пересчет положения центра кластера. Условие остановки алгоритма – точки перестали перемещаться между кластерами, либо число итераций достигло заданного максимального значения.

Формат входного файла: в каждой отдельной строке координаты точки x, y

Формат выходного файла: каждая отдельная строка – описание кластера, x\_c, y\_c – координаты центра кластера, далее перечислены все точки, входящие в кластер

((x\_c, y\_c) ((x, y), (x,y), … ))

1. Реализовать алгоритм поиска в ширину на графе. Граф представлен в виде списка смежности – каждой вершине соответствует список, состоящий из вершин, до которых есть ребра, выходящие из данной вершины. Результат представить в виде списка вершин в порядке их обхода алгоритмом.

# КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Дайте определение Apache Spark.
2. Раскройте преимущества Apache Spark перед Hadoop MapReduce.
3. Перечислите модули, из которых состоит Apache Spark.
4. Опишите основные этапы установки Apache Spark.
5. Приведите команду для запуска Spark Shell.
6. Дайте определение RDD.
7. Перечислите основные типы операций над RDD.
8. Перечислите основные трансформации, производимые над RDD.
9. Перечислите основные действия, выполняемые над RDD.
10. Приведите команды для загрузки данных в RDD.

# ФОРМА ОТЧЕТА ПО ДОМАШНЕЙ РАБОТЕ

На выполнение домашней работы отводится 10 часов. Номер варианта студенту выдается преподавателем. Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания (вариант), этапы выполнения работы (со скриншотами), результаты выполнения работы. выводы.

# ОСНОВНАЯ ЛИТЕРАТУРА

1. Федин Ф.О. Анализ данных. Часть 1. Подготовка данных к анализу [Электронный ресурс] : учебное пособие / Ф.О. Федин, Ф.Ф. Федин.
   * Электрон. текстовые данные. — М. : Московский городской педагогический университет, 2012. — 204 c. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/26444.html>
2. Федин Ф.О. Анализ данных. Часть 2. Инструменты Data Mining [Электронный ресурс] : учебное пособие / Ф.О. Федин, Ф.Ф. Федин.
   * Электрон. текстовые данные. — М. : Московский городской педагогический университет, 2012. — 308 c. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/26445.html>
3. Чубукова, И.А. Data Mining [Электронный ресурс] : учеб. пособие

— Электрон. дан. — Москва : , 2016. — 470 с. — Режим доступа: https://e.lanbook.com/book/100582. — Загл. с экрана.

1. Воронова Л.И. Big Data. Методы и средства анализа [Электронный ресурс] : учебное пособие / Л.И. Воронова, В.И. Воронов. — Электрон. текстовые данные. — М. : Московский технический университет связи и информатики, 2016. — 33 c. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/61463.html>

# ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

1. Волкова Т.В. Разработка систем распределенной обработки данных [Электронный ресурс] : учебно-методическое пособие / Т.В. Волкова, Л.Ф. Насейкина. — Электрон. текстовые данные. — Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2012. — 330 c. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/30127.html>
2. Кухаренко Б.Г. Интеллектуальные системы и технологии [Электронный ресурс] : учебное пособие / Б.Г. Кухаренко. — Электрон. текстовые данные. — М. : Московская государственная академия водного транспорта, 2015. — 116 c. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/47933.html>
3. Воронова Л.И. Интеллектуальные базы данных [Электронный ресурс] : учебное пособие / Л.И. Воронова. — Электрон. текстовые данные. — М. : Московский технический университет связи и информатики, 2013. — 35 c. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/63324.html>
4. Николаев Е.И. Базы данных в высокопроизводительных информационных системах [Электронный ресурс] : учебное пособие / Е.И. Николаев. — Электрон. текстовые данные. — Ставрополь: Северо-Кавказский федеральный университет, 2016.

— 163 c. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/69375.html>

# Электронные ресурсы:

1. <http://hadoop.apache.org/>(англ.)
2. https://spark.apache.org/ (англ.)