

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**

**MAPREDUCE**

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

Калуга - 2018

УДК 004.62  
ББК 32.972.5  
Б435

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий и прикладной математики».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий и прикладной математики» (ФН1-КФ) протокол № 6 от «12» января 2018 г.


Зав. кафедрой ФН1-КФ  д.ф.-м.н., профессор Б.М. Логинов

- Методической комиссией факультета ФНК протокол № 1 от «30» 01 2018 г.

Председатель методической комиссии факультета ФНК  к.х.н., доцент К.Л. Анфилов

- Методической комиссией КФ МГТУ им.Н.Э. Баумана протокол № 1 от «06» 02 2018 г.

Председатель методической комиссии КФ МГТУ им.Н.Э. Баумана

 д.э.н., профессор О.Л. Перерва

Рецензент:



к.т.н., зав. кафедрой ЭИУ2-КФ

 И.В. Чухраев

Авторы

к.ф.-м.н., доцент кафедры ФН1-КФ

ассистент кафедры ФН1-КФ

 Ю.С. Белов  
 С.С. Гришунов

#### Аннотация

Методические указания по выполнению лабораторной работы по курсу «Технологии обработки больших данных» содержат краткое описание принципа MapReduce и всех его этапов работы, а также пример решения задачи подсчета количества слов в файлах, использующий принцип MapReduce.

Предназначены для студентов 4-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

© Калужский филиал МГТУ им. Н.Э. Баумана, 2018 г.

© Ю.С. Белов, С.С. Гришунов, 2018 г.

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ .....	4
ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ.....	5
КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ .....	6
ПРИМЕР MAPREDUCE ЗАДАЧИ .....	9
ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ .....	15
ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ.....	15
ВАРИАНТЫ ЗАДАНИЙ.....	15
КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ .....	17
ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ .....	17
ОСНОВНАЯ ЛИТЕРАТУРА.....	18
ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА .....	18

## **ВВЕДЕНИЕ**

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «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», содержат краткое описание принципа MapReduce и всех его этапов работы, а также примеры решения задач и задание на выполнение лабораторной работы.

Методические указания составлены для ознакомления студентов с подходом MapReduce для обработки больших данных. Для выполнения лабораторной работы студенту необходимы минимальные знания по программированию на высокоуровневом языке программирования (Java, Python или др.).

## **ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ**

Целью выполнения лабораторной работы является формирование практических навыков использования парадигмы MapReduce для обработки больших данных.

Основными задачами выполнения лабораторной работы являются:

1. Изучить подход MapReduce.
2. Изучить принципы работы Hadoop MapReduce.
3. Получить практические навыки реализации MapReduce задач.
4. Уметь обрабатывать большие текстовые файлы с помощью MapReduce.

Результатами работы являются:

- Входные файлы с данными
- MapReduce-программа, обрабатывающая данные согласно варианту задания
- Выходные файлы с результатами вычислений
- Подготовленный отчет

## КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

Обработка больших данных стандартными приемами приведет к ряду проблем:

1. Объем обрабатываемых данных ограничен памятью сервера.
2. Невозможно распараллелить обработку данные на нескольких серверах.

Можно сформулировать основные принципы работы с большими данными:

**1. Горизонтальная масштабируемость.** Поскольку данных может быть сколь угодно много – любая система, которая подразумевает обработку больших данных, должна быть расширяемой.

**2. Отказоустойчивость.** Принцип горизонтальной масштабируемости подразумевает, что машин в кластере может быть много. Например, Надоор-кластер Yahoo имеет более 42000 машин. Это означает, что часть этих машин будет гарантированно выходить из строя. Методы работы с большими данными должны учитывать возможность таких сбоев и переживать их без каких-либо значимых последствий.

**3. Локальность данных.** В больших распределённых системах данные распределены по большому количеству машин. Если данные физически находятся на одном сервере, а обрабатываются на другом – расходы на передачу данных могут превысить расходы на саму обработку. Поэтому одним из важнейших принципов проектирования BigData-решений является принцип локальности данных – по возможности обрабатываем данные на той же машине, на которой они хранятся.

Все современные средства работы с большими данными так или иначе следуют этим трём принципам.

**MapReduce** – это модель распределенной обработки данных, предложенная компанией Google для обработки больших объёмов данных на компьютерных кластерах.

MapReduce предполагает, что данные организованы в виде некоторых записей. Обработка данных происходит в 3 стадии (рис. 1):

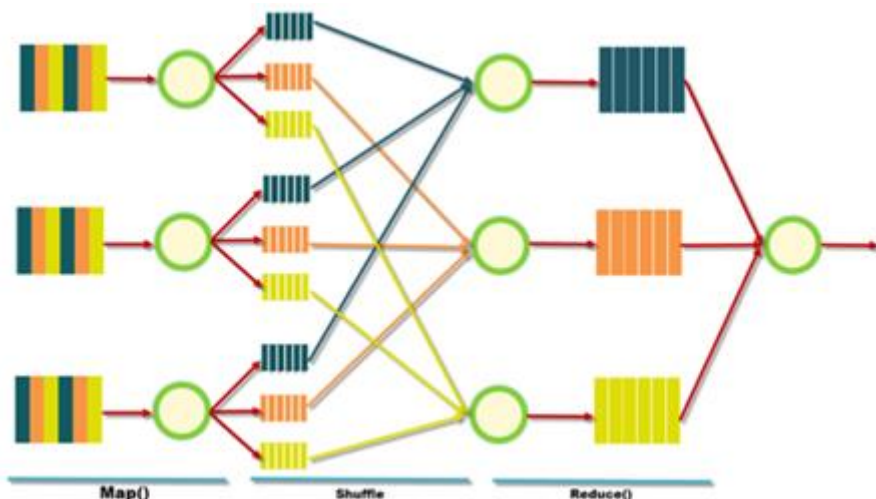


Рис. 1 – Стадии работы MapReduce задачи

Рассмотрим каждую из стадий:

**1. Стадия Map.** На этой стадии данные преобразовываются при помощи функции `map()`, которую определяет пользователь. Работа этой стадии заключается в преобразовке и фильтрации данных. Работа очень похожа на операцию `map` в функциональных языках программирования – пользовательская функция применяется к каждой входной записи.

Функция `map()` примененная к одной входной записи и выдаёт множество пар ключ-значение. Данная функция может выполняться на каждом сервере независимо и параллельно над тем набором данным, которым оперирует данный сервер.

**2. Стадия Shuffle.** Проходит незаметно для пользователя. В этой стадии вывод функции `map` «разбивается по корзинам» – каждая корзина соответствует одному ключу вывода стадии `map`. В дальнейшем эти корзины послужат входом для `reduce`.

**3. Стадия Reduce.** Каждая «корзина» со значениями, сформированная на стадии `shuffle`, попадает на вход функции `reduce()`.

Функция `reduce` задаётся пользователем и вычисляет финальный результат для отдельной «корзины». Множество всех значений, возвращённых функцией `reduce()`, является финальным результатом MapReduce-задачи.

Свойства MapReduce задач:

1. Все запуски функции map работают независимо и могут работать параллельно, в том числе на разных машинах кластера.
2. Все запуски функции reduce работают независимо и могут работать параллельно, в том числе на разных машинах кластера.
3. Shuffle внутри себя представляет параллельную сортировку, поэтому также может работать на разных машинах кластера. Пункты 1-3 позволяют выполнить принцип [горизонтальной масштабируемости](#).
4. Функция map, как правило, применяется на той же машине, на которой хранятся данные – это позволяет снизить передачу данных по сети (принцип [локальности данных](#)).



## ПРИМЕР MAPREDUCE ЗАДАЧИ

Классическим примером MapReduce задачи является подсчет слов в большом количестве документов (Word count).

Задача формулируется следующим образом: имеется большой корпус документов. Задача – для каждого слова, хотя бы один раз встречающегося в корпусе, посчитать суммарное количество раз, которое оно встретилось в корпусе.

Входной записью для MapReduce задачи будет являться каждый отдельный документ. Функция [map](#) обрабатывает каждое слово в документе и превращает один входной документ в набор пар (word, 1). Стадия [shuffle](#) прозрачно для пользователя группирует данные пары по словам (word, [1,1,1,1,1,1]). Функция [reduce](#) суммирует единицы, возвращая финальный результат для каждого слова (word, count).

### Реализация Word Count на Java

Для реализации MapReduce задачи подсчета слов будем использовать систему Hadoop, установленную и настроенную в лабораторной работе №1. Hadoop имеет встроенный MapReduce framework, работающий с файловой системой HDFS.

Для запуска MapReduce задачи нужно указать как минимум расположение входных/выходных файлов, задать функцию map и reduce, реализовав соответствующие интерфейсы. Эти и другие параметры являются конфигурацией задачи (job). После конфигурации задача передается Менеджеру ресурсов (Resource Manager), который отвечает за распределение и планирование выполнения задач по всем узлам.

```
public class WordCount {  
  
    public static class TokenizerMapper  
        extends Mapper<  
            Object,           // Input key Type  
            Text,             // Input value Type  
            Text,             // Output key Type  
            IntWritable>      // Output value Type  
  
    {  
        private final static IntWritable one = new IntWritable(1);  
        private Text word = new Text();
```

```

public void map(Object key, Text value, Context context)
    throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
    }
}
}

```

```

public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
        Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}

```

```

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

Класс Mapper выполняет функцию Map в MapReduce задаче. Он преобразует входные пары ключ-значение в промежуточные пары ключ-значение. Типы промежуточных записей могут отличаться от входных пар ключ-значение.

Реализация Mapper передается задаче через вызов метода `Job.setMapperClass`. Затем фреймворк самостоятельно вызывает функцию map для каждой входной записи. Преобразованные пары ключ-значение передаются на следующий этап вызовом метода `context.write`.

Все преобразованные пары ключ-значение группируются фреймворком и передаются в Reducer. Пользователь может контролировать группировку, определив класс `Comparator` и установив его вызовом `Job.setGroupingComparatorClass`.

Также дополнительно можно определить класс `Combiner`, который будет выполнять локальную группировку промежуточных пар. `Combiner` помогает снизить количество информации, передаваемое от Mapper к Reducer.

Функция map обрабатывает по одному текстовому файлу. С помощью объекта `StringTokenizer` текст разделяется на токены, разделенные пробельными символами. Для каждого токена возвращается пара ключ-значение (`word, 1`).

В качестве тестового примера создадим 2 файла:

file01:

Hello World Bye World

file02:

Hello Hadoop Goodbye Hadoop

Для тестового примера первый вызов map вернет:

(Hello, 1)

(World, 1)

(Bye, 1)

(World, 1)

Второй вызов map вернет:

(Hello, 1)

(Hadoop, 1)

(Goodbye, 1)

(Hadoop, 1)

Затем результат работы функций `map` передается объекту `Combiner`. `Combiner` выполняется локально для каждого файла и используется для локальной группировки ключей. В данном случае `Combiner` и `Reducer` будут одинаковы. `Reducer` и `Combiner` суммируют значения, которые и являются количеством появления слов в тексте. `Combiner` складывает значения в рамках одного файла, результатом его работы будут:

Для первого файла:

(Bye, 1)  
(Hello, 1)  
(World, 2)

Для второго файла:

(Goodbye, 1)  
(Hadoop, 2)  
(Hello, 1)

Реализация `Reducer` передается задаче через вызов метода `Job.setReducerClass`. Затем фреймворк самостоятельно вызывает функцию `reduce` для каждой сгруппированной по ключу паре. Результат работы `Reducer` является результатом работы всей `MapReduce` задачи:

(Bye, 1)  
(Goodbye, 1)  
(Hadoop, 2)  
(Hello, 2)  
(World, 2)

Метод `main` определяет параметры конфигурации задачи, такие как пути ввода/вывода (передаются через командную строку), типы ключей и значений, форматы ввода/вывода. Затем вызывается метод `job.waitForCompletion`, который ожидает завершения выполнения задачи.

### **Запуск MapReduce задачи**

Скомпилируем файл `wordcount.java` в `jar` архив `wc.jar` командой:

```
bin/hadoop com.sun.tools.javac.Main WordCount.java  
jar cf wc.jar WordCount*.class
```

Создадим входные файлы в локальной файловой системе и перенесем их в HDFS:

```
bin/hadoop fs -copyFromLocal /input/file01
bin/hadoop fs -copyFromLocal /input/file02
```

Запускаем приложение из wc.jar, в качестве параметров передаем входной и выходной путь:

```
bin/hadoop jar wc.jar WordCount/user/hduser/wordcount/input
/user/hduser/wordcount/output
```

Результат работы приложения можно найти в user/hduser/wordcount/output.

```
bin/hadoop fs -cat /user/joe/wordcount/output/part-r-00000
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

## Streaming

Несмотря на то, что Hadoop реализован на языке Java, существует возможность реализовывать MapReduce приложения на любом языке, позволяющем использовать стандартный консольный ввод-вывод.

Для этого необходимо использовать специальное приложение java-streaming:

```
$HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/hadoop-streaming.jar
-input myInputDirs \
-output myOutputDir \
-mapper org.apache.hadoop.mapred.lib.IdentityMapper \
-reducer /bin/wc
```

Команда Hadoop jar позволяет запустить в Hadoop jar файл. В качестве параметра команды необходимо указать имя jar файла, в роли которого выступает hadoop-streaming.jar. Далее в команде в виде ключей необходимо передать директорию с входными файлами в hdfs, директорию, в которую будет выводиться результат вычислений, а также скрипты, в которых описаны mapper и reducer.

## ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Выполнить задание с помощью подхода MapReduce согласно варианту. В качестве входных текстовых файлов можно использовать книги в txt формате из библиотеки Project Gutenberg: <https://www.gutenberg.org>.

Список стоп-слов: <http://xpo6.com/wp-content/uploads/2015/01/stop-word-list.csv>

## ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ

Программа может быть реализована на любом языке высокого уровня, для которого существует поддержка работы с HDFS (Java, Python, Scala или др.). Имена файлов должны передаваться приложению в качестве ключей при вызове в терминале.

## ВАРИАНТЫ ЗАДАНИЙ

1. Подсчитать количество строк в файле. Результат должен быть сохранен в файле в виде:

`file_name lines_count`

2. Подсчитать количество появлений каждого буквенного символа в файле. Подсчет должен быть регистро-зависимым (т.е. буквы «а» и «А» считаются разными). Результат должен быть сохранен в файле в виде:

`((a 484) (b 95) (c 187) ...)`

3. Реализовать поиск слова в нескольких файлах. Результат должен содержать номера всех строк в каждом файле, в которых появляется заданное слово. Сохранить результат в файл в виде:

`(word (7@file1 46@file1 52@file2 63@file2 ...))`

4. Построить индекс файла. Для каждого слова в файле результат должен содержать номера всех строка, в которых появляется данное слово. Индекс должен быть регистро-независимым. Результат должен быть сохранен в файле в виде:

((word1 (1 42 58)), (word2 (34, 55, 776, 3456), ...))

5. Модифицировать программу подсчета слов WordCount для удаления стоп-слов, знаков пунктуации и цифр. Список стоп-слов должен находиться в отдельном файле.

6. Модифицировать программу подсчета слов WordCount для подсчета слов, начинающихся с заданной подстроки. Из результата должны быть удалены стоп-слова.

7. Модифицировать программу подсчета слов WordCount. Результат должен содержать 100 самых часто встречающихся слов. Из результата должны быть удалены стоп-слова.

8. Построить обратный индекс для файлов. Обратный индекс для каждого слова содержит список имен файлов, в которых оно встречается, и количество появлений слова в каждом файле. Результат должен быть сохранен в файле в виде:

(word1 (file1 42), (file2 25)), (word2 (file1, 55)), ...)

9. Реализовать умножение матриц.

Входной файл имеет формат:

имя\_матрицы, строка, столбец, значение.

Результат выполнения стадии Map представить в виде пар ключ-значение, где ключ – индексы элемента вычисляемой матрицы, а значение – список значений, необходимых для вычисления данного элемента.

10. Подсчитать средний рейтинг фильма. Входной файл имеет формат:

userId, movieId, rating, timestamp.

Результат должен быть сохранен в файле в формате:

movieId, av\_rating

Входной файл: [rating.csv](#)



## **КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ**

1. Сформулируйте основные принципы работы с большими данными.
2. Перечислите основные стадии решения MapReduce задачи.
3. Опишите роль стадии Map.
4. Опишите роль стадии Shuffle.
5. Опишите роль стадии Reduce.
6. Как задать в java-приложении mapper-класс и reducer-класс.
7. Приведите команды для задания входных и выходных директорий файлов java MapReduce приложения.
8. Опишите назначение Combiner класс.
9. Опишите процесс компиляции и запуска java MapReduce приложения.
10. Приведите команду для запуска MapReduce приложения, написанного на каком-либо языке, отлично от java.

## **ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ**

На выполнение лабораторной работы отводится 3 занятия (6 академических часов: 5 часов на выполнение и сдачу лабораторной работы и 1 час на подготовку отчета).

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 с. — 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 с. — 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.
4. 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 с. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/61463.html>
5. Юре, Л. Анализ больших наборов данных [Электронный ресурс] / Л. Юре, Р. Ананд, Д.У. Джеффри. — Электрон. дан. — Москва : ДМК Пресс, 2016. — 498 с. — Режим доступа: <https://e.lanbook.com/book/93571>. — Загл. с экрана.

## ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

6. 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 с. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/30127.html>

7. 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 с. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/47933.html>
8. 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 с. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/63324.html>
9. 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 с. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/69375.html>

### **Электронные ресурсы:**

10. <http://hadoop.apache.org/> (англ.)