

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

ЦЕПОЧКИ MAPREDUCE ЗАДАЧ. СРАВНЕНИЕ ДОКУМЕНТОВ

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.

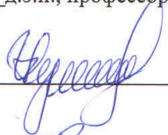
Председатель методической комиссии факультета ФНК  к.х.н., доцент К.Л. Анфилов

- 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 построения цепочек MapReduce задач. Рассмотрен пример построения цепочки MapReduce задач для нахождения вероятности взаимного появления двух слов в предложении.

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 ЦЕПОЧКИ MAPREDUCE ЗАДАЧ ДЛЯ НАХОЖДЕНИЯ ВЕРОЯТНОСТИ ВЗАИМНОГО ПОЯВЛЕНИЯ ДВУХ СЛОВ В ПРЕДЛОЖЕНИИ.....	8
ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ	13
ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ.....	13
ВАРИАНТЫ ЗАДАНИЙ.....	13
КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ	18
ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ	18
ОСНОВНАЯ ЛИТЕРАТУРА.....	19
ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА	19

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 описание принципа построения MapReduce задач, 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 цепочек MapReduce для решения сложных задач обработки больших данных.

Основными задачами выполнения лабораторной работы являются:

1. Получить навыки реализации цепочки MapReduce задач.
2. Изучить интерфейс Hadoop MapReduce.
3. Изучить алгоритмы анализа, сравнение текстовых документов
4. Получить практические навыки обработки и анализа текстовых данных

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

MapReduce состоит из стадий Map, Shuffle и Reduce. Как правило, в практических задачах самой тяжёлой оказывается стадия Shuffle, так как на этой стадии происходит сортировка данных. На самом деле существует ряд задач, в которых можно обойтись только стадией Map. Вот примеры таких задач:

- Фильтрация данных (например, «Найти все записи с IP-адреса 123.123.123.123» в логах web-сервера);
- Преобразование данных («Удалить колонку в csv-логах»);
- Загрузка и выгрузка данных из внешнего источника («Вставить все записи из лога в базу данных»).

Такие задачи решаются при помощи Map-Only. При создании Map-Only задачи в Hadoop нужно указать нулевое количество reducer'ов.

Как было сказано выше, обычно самая трудоемкая стадия при выполнении Map-Reduce задачи – это стадия shuffle. Происходит это потому, что промежуточные результаты (выход mapper'а) записываются на диск, сортируются и передаются по сети. Однако существуют задачи, в которых такое поведение кажется не очень разумным. Например, в той же задаче подсчёта слов в документах можно предварительно агрегировать результаты выходов нескольких mapper'ов на одном узле map-reduce задачи, и передавать на reducer уже просуммированные значения по каждой машине.

В Hadoop для этого можно определить комбинирующую функцию, которая будет обрабатывать выход части mapper'ов. Комбинирующая функция очень похожа на reduce – она принимает на вход выход части mapper'ов и выдаёт агрегированный результат для этих mapper'ов, поэтому очень часто reducer используют и как combiner. Важное отличие от reduce – на комбинирующую функцию попадают не все значения, соответствующие одному ключу, а только значения, полученные на данном сервере.

Более того, Hadoop не гарантирует того, что комбинирующая функция вообще будет выполнена для выхода mapper'а. Поэтому

комбинирующая функция не всегда применима, например, в случае поиска медианного значения по ключу (т.к. чтобы найти медиану необходимо отсортировать все значения в порядке возрастания и выбрать элемент, находящийся посередине, а для того чтобы ключи отсортировать надо знать все значения, относящиеся к ключу). Тем не менее, в тех задачах, где комбинирующая функция применима, её использование позволяет добиться существенного прироста к скорости выполнения MapReduce-задачи.

Бывают ситуации, когда для решения задачи одним MapReduce не обойтись. Например, рассмотрим немного видоизмененную задачу WordCount: имеется набор текстовых документов, необходимо посчитать, сколько слов встретилось от 1 до 1000 раз в наборе, сколько слов от 1001 до 2000, сколько от 2001 до 3000 и так далее.

Для решения нам потребуется 2 MapReduce job'a:

1. Видоизменённый wordcount, который для каждого слова рассчитает, в какой из интервалов оно попало;
2. MapReduce, подсчитывающий, сколько раз в выходе первого MapReduce встретился каждый из интервалов.

Для того, чтобы выполнить последовательность MapReduce-задач на Hadoop, достаточно просто в качестве входных данных для второй задачи указать папку, которая была указана в качестве output для первой и запустить их по очереди.

На практике цепочки MapReduce-задач могут представлять собой достаточно сложные последовательности, в которых MapReduce-задачи могут быть подключены как последовательно, так и параллельно друг другу, могут отсутствовать как Reduce, так и Map стадии.

ПРИМЕР ЦЕПОЧКИ MAPREDUCE ЗАДАЧ ДЛЯ НАХОЖДЕНИЯ ВЕРОЯТНОСТИ ВЗАИМНОГО ПОЯВЛЕНИЯ ДВУХ СЛОВ В ПРЕДЛОЖЕНИИ

Подсчитаем вероятность взаимного появления двух слов в предложении. Если предложение состоит из слов “А В С”, то взаимно появляющимися парами будут:

(А, В)

(А, С)

(В, А)

(В, С)

(С, А)

(С, В)

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

$$PMI(x, y) = \ln \frac{p(x, y)}{p(x)p(y)}$$

где $p(x, y)$ – частота взаимного появления двух слов, $p(x)$, $p(y)$ – частота появления слова в отдельности.

Для реализации будет использоваться 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 также должны быть удалены стоп-слова.

Данную задачу невозможно решить в рамках [одной стадии MapReduce](#). Рассмотрим первый mapper (приведена реализация на языке python):

```
def process(sentence, separator):  
    words = splitter.split(sentence);  
    word_c = 0  
    pair_c = 0  
    i = 0  
    while i < len(words):
```



```

    if words[i] == "":
        i += 1
        continue
    j = 0
    word_c += 1
    print '%s%s%d' % (words[i], separator, 1)
    while j < len(words):
        if i != j and words[j] != "":
            pair_c += 1
            print '%s!%s%s%d' % (words[i], words[j], separator, 1)
            j += 1
        i += 1
    return word_c, pair_c

def main(separator='\t'):
    stop_words = read_stop_words('stop-word-list.csv')
    word_c = 0
    pair_c = 0
    for sentences in read_input(sys.stdin):
        for sentence in sentences:
            wc, pc = process(sentence, separator)
            word_c += wc
            pair_c += pc
    print '!%d%s%d' % (word_c, separator, pair_c)

```

Скрипт принимает на вход файл и возвращает построчно слова и встреченные пары в строке. Reducer вычисляет вероятность нахождения слова в данном файле:

```

separator = '\t'
current_word = None
current_count = 0
word = None
last_c = 0
curr_c = 0

```

```

word_c = 0
pair_c = 0

for line in sys.stdin:
    line = line.strip()
    if line[0] == '!':
        elements = line.split('!')
        elements = elements[1].split(separator)
        word_c += int(elements[0])
        pair_c += int(elements[1])
        continue
    elements = line.split(separator)
    key = elements[0].split('!')
    word = elements[0]
    count = elements[1]
    if len(key) == 1:
        curr_c = word_c
    elif len(key) == 2:
        curr_c = pair_c
    else:
        continue

    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            p = 1.0 * current_count / last_c
            print '%s%s%s' % (current_word, separator, p)
            current_count = count
            current_word = word
            last_c = curr_c

if current_word == word:
    p = 1.0 * current_count / last_c
    print '%s%s%s' % (current_word, separator, p)

```

Дальнейшее использование Mapper'ов не позволит достичь результата, следовательно, будут использоваться только Reducer'ы. Следующий Reducer вычисляет вероятность взаимного появления слов в файле:

```
tab = '\t'
separator = '!'
p1 = 0
w1 = None
w2 = None

for line in sys.stdin:
    line = line.strip()
    elements = line.split(tab)
    key = elements[0].split(separator)
    try:
        if len(key) == 1:
            w1 = key[0]
            w2 = None
            p1 = float(elements[1])
        elif len(key) == 2:
            w1 = key[0]
            w2 = key[1]
            p12 = float(elements[1])
        else:
            continue
    except ValueError:
        continue

    if w2:
        if p1 == 0:
            continue
        print '%s%s%s%s%s' % (w2, separator, w1, tab, p12 / p1)
    else:
        print '%s%s%s' % (w1, tab, p1)
```

Последний Reducer вычисляет метрику метрику PMI:

```
tab='\t'
```

```

separator = '!'
p1 = 0
w1 = None
w2 = None

for line in sys.stdin:
    line = line.strip()
    elements = line.split(tab)
    key = elements[0].split(separator)
    if len(key) == 1:
        w1 = key[0]
        w2 = None
        p1 = float(elements[1])
    elif len(key) == 2:
        w1 = key[0]
        w2 = key[1]
        p12 = float(elements[1])
    else:
        continue

    if w2:
        print 'PMI(%s,%s)=%s%s' % (w1, w2, tab, math.log(p12 / p1))

```

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Выполнить задание с помощью цепочки MapReduce задач 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.).

ВАРИАНТЫ ЗАДАНИЙ

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.

Например, для файлов:

File1:

He put some sugar into his coffee, as he always did. However, today his coffee was too sweet.

File2:

He had sugar in his coffee, though he is diabetic.

результатом будет:

2	He
1	sugar
1	his
1	coffee

Из результата должны быть удалены стоп-слова, которые находятся в отдельном файле. Результат должен содержать только 20 самых часто встречающихся слов в обоих файлах, отсортированных в порядке убывания.

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

Метрика $TF \cdot IDF$ для слова s в документе d вычисляется:

$$TF \cdot IDF = (1 + \ln TF) * \ln \frac{N}{IDF}$$

где N – общее количество документов, TF – количество встречаения 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 $TF \cdot IDF$. 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 $TF \cdot IDF$ для слов, присутствующих в запросе.

Стадия 5:

Отбор 3 самых подходящих документов.

3. Подсчитать вероятность взаимного появления двух слов в предложении. Если предложение состоит из слов “А В С”, то взаимно появляющимися парами будут:

(А, В)

(А, С)

(В, А)

(В, С)

(С, А)

(С, В)

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

$$PMI(x, y) = \ln \frac{p(x, y)}{p(x)p(y)}$$

где $p(x, y)$ – частота взаимного появления двух слов, $p(x)$, $p(y)$ – 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, содержащий все слова с которыми есть взаимное появление слова a в файле(ключ – слово b , а значение – $PMI(a, b)$).

Из результата должны быть удалены стоп-слова.

4. Реализовать алгоритм PageRank. Входные данные – ориентированный граф (V, E) , где вершины V представляют web-страницу, а ребра E – ссылки на страницы. Граф представлен в виде списка смежности – каждой странице соответствует список, состоящий из страниц, на которые есть ссылки с данной страницы. Рассчитать значение PageRank для каждой страницы, для расчета использовать как минимум 10 итераций.

$$PR(v) = \frac{1-d}{N} + d \sum_{i=1}^n \frac{PR(u_i)}{C(u_i)}$$

где $u_1, u_2 \dots$ – страницы, на которые есть ссылки со страницы v ,

$C(u_i)$ – Количество ребер, выходящих из вершины u_i ,

d – константа, принять равной 0,85

N – общее количество страниц.

Начальное значение PR для всех страниц задать равным 1.

Вывести результат в отсортированном по убыванию значения PR списке страниц.

Во входном файле первое число в каждой строке – id страницы, далее перечислены id страниц, на которые есть ссылка с исходной страницы.

Входной файл:

<http://lintool.github.io/Cloud9/docs/exercises/sample-medium.txt>

5. 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.

6. Реализовать алгоритм поиска в глубину 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.

7. Подсчитать средний рейтинг фильма. Входной файл рейтингов имеет формат:

userId, movieId, rating, timestamp

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

Исходные файлы:

[rating.csv](#)

[movies.csv](#)

8. Подсчитать средний рейтинг фильма. Входной 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](#)

9. 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](#)

10. 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), ...))

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Перечислите стадии, из которых состоит MapReduce.
2. Приведите наиболее вычислительно трудную стадию MapReduce.
3. Раскройте значение термина Map-Only задачи.
4. Дайте определение комбинирующей функции.
5. Приведите примеры задач, для которых применение комбинирующих функций поможет повысить производительность.
6. Приведите примеры задач, для которых неприменимы комбинирующие функции.
7. Приведите пример задач, которые невозможно решить одной MapReduce стадией.
8. Раскройте область применения цепочек MapReduce задач
9. Приведите команды для выполнения последовательности MapReduce-задач.

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.)