Министерство науки и высшего образования 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)

ЦЕПОЧКИ MAPREDUCE ЗАДАЧ. СРАВНЕНИЕ ДОКУМЕНТОВ

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

Калуга – 2023

ОГЛАВЛЕНИЕ

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

Бывают ситуации, когда для решения задачи одним MapReduce не обойтись. Многие из них можно решить только разбив над подзадачи, каждую из которых будет решать свой Mapper и Reducer, т.е. схематично решение задачи будет выглядеть следующим образом:

Map1 → Reduce1 → Map2 → Reduce2 → Map3…

Существует несколько способов создания цепочек MapReduce задач.

Самым тривиальным способом будет создание каскада задач. Для этого достаточно последовательно создать два объекта класса *JobConf* (например, *job1* и *job2*), настроить пути входных и выходных данных и запустить задачи следующим образом:

```
JobClient.run(job1);
JobClient.run(job2);
```

Можно изменить подход и поступить другим способом. Для начала так же, как и в первом случае нужно последовательно создать два объекта класса *JobConf*, но не выполнять задачи, используя *JobClient.run()*. Вместо этого необходимо создать два объекта класса *Job* и в качестве параметра указать объекты *JobConf*:

```
Job job1=new Job(jobconf1);
Job job2=new Job(jobconf2);
```

После этого управление задачами происходит с помощью класса *JobControl*:

```
JobControl jbcntrl=new JobControl("jbcntrl");
jbcntrl.addJob(job1);
jbcntrl.addJob(job2);
job2.addDependingJob(job1);
jbcntrl.run();
```

Однако создание каскада задач подойдет только для самых простых случаев. Для более сложных задач в Hadoop существуют классы **ChainMapper** и **ChainReducer**.

ChainMapper в Hadoop

ChainMapper – один из предопределенных классов MapReduce в Hadoop. Класс ChainMapper позволяет использовать несколько классов Mapper в одной задаче. Классы Mapper вызываются в цепочке, где выходные данные первого Mapper становятся входными данными второго, и так до последнего Mapper. Выходные данные последнего Mapper будут записаны в выходной файл задачи.

Для добавления задач в ChainMapper используется метод *addMapper()*.

ChainReducer в Hadoop

Класс ChainReducer позволяет связывать несколько классов Mapper внутри задачи, выполняемой Reducer.

Для каждой записи, выводимой Reducer, классы Mapper вызываются по цепочке. Выходные данные Reducer становятся входными данными первого Mapper, а выходные данные этого первого Mapper становятся входными данными второго, и так до последнего Mapper. Выходные данные последнего Mapper будут записаны в файл в качестве результата выполнения задачи.

Для смены Reducer в цепочке заданий используется метод *setReducer()*.

Таким образом, возможны следующие варианты создания цепочек MapReduce задач:

1. Цепочка map задач, выполняемых с помощью ChainMapper.
2. Изменение Reducer в цепочке задач с помощью ChainReducer.
3. Добавление цепочки задач с помощью ChainReducer.

ПРИМЕР ЦЕПОЧКИ MAPREDUCE ЗАДАЧ

Пример №1. Цепочки mapreduce задач с использованием chainmapper и chainreducer

Предположим, что у нас есть входной файл следующего формата:

Item1 345 zone-1
Item1 234 zone-2
Item3 654 zone-2
Item2 231 zone-3

Файл содержит столбцы наименования товара, количества продаж и зоны продаж (разделенные табуляцией). Нужно получить общее количество продаж по каждому товару для зоны-1.

Для решения данной задачи можно поступить следующим образом. Создать два Mapper: первый будет отвечать за считывание всех записей, а второй - за фильтрацию записей, для получения только тех, которые сделаны для зоны-1. В Reducer будет происходить получение общего количества продаж (суммирование) для отфильтрованных товаров.

Исходный код на языке Java:

```
package org.example;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import
org.apache.hadoop.mapreduce.lib.chain.ChainMapper;
import
org.apache.hadoop.mapreduce.lib.chain.ChainReducer;
```

```
import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import
org.apache.hadoop.mapreduce.lib.map.InverseMapper;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat
;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class Sales extends Configured implements Tool {
    // Первый Mapper
    public static class CollectionMapper extends
Mapper<LongWritable, Text, Text, Text>{
        private Text item = new Text();

        public void map(LongWritable key, Text value,
Context context) throws IOException,
InterruptedException {
            // Делим строку по знаку табуляции
            String[] salesArr =
value.toString().split("\t");
            item.set(salesArr[0]);

            // Записываем значения количества продаж и
зон
            context.write(item, new Text(salesArr[1] +
"," + salesArr[2]));
        }
    }

    // Второй Mapper
    public static class FilterMapper extends
Mapper<Text, Text, Text, IntWritable>{
        public void map(Text key, Text value, Context
context) throws IOException, InterruptedException {
            String[] recordArr =
value.toString().split(",");

            // Фильтруем записи
            if(recordArr[1].equals("zone-1")) {
```

```
                Integer sales =
Integer.parseInt(recordArr[0]);
                context.write(key, new
IntWritable(sales));
            }
        }
    }

    // Reducer
    public static class TotalSalesReducer extends
Reducer<Text, IntWritable, Text, IntWritable>{
        public void reduce(Text key,
Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws
Exception {
        int exitFlag = ToolRunner.run(new Sales(),
args);
        System.exit(exitFlag);
    }

    @Override
    public int run(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Sales");
        job.setJarByClass(getClass());

        // Создание цепочки MapReduce задач
        Configuration mapConf1 = new
Configuration(false);
        ChainMapper.addMapper(job,
CollectionMapper.class, LongWritable.class, Text.class,
Text.class, Text.class,  mapConf1);
```

```
        Configuration mapConf2 = new
Configuration(false);
        ChainMapper.addMapper(job, FilterMapper.class,
Text.class, Text.class, Text.class, IntWritable.class,
mapConf2);

        Configuration reduceConf = new
Configuration(false);
        ChainReducer.setReducer(job,
TotalSalesReducer.class, Text.class, IntWritable.class,
Text.class, IntWritable.class, reduceConf);

        ChainReducer.addMapper(job,
InverseMapper.class, Text.class, IntWritable.class,
IntWritable.class, Text.class, null);

        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new
Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
Path(args[1]));

        return job.waitForCompletion(true) ? 0 : 1;
    }
  }
```

Обратите внимание, что в качестве добавляемого к ChainReducer Mapper используется предопределенный класс InverseMapper. Он необходим для того, чтобы поменять местами ключи и значения в конечном результате.

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.

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. У нас есть файл с информацией о продажах различных книг в разных книжных магазинах. Каждая строка файла содержит данные о названии книги, количестве проданных экземпляров и городе продажи (разделенные пробелами):

Book1 50 City-A
Book2 30 City-B
Book1 20 City-B
Book3 40 City-A

Необходимо определить общее количество проданных книг каждого названия в *City-A*.

2. 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.

3. Допустим, у нас есть файл с информацией о продажах различных видов музыкальных инструментов в разных музыкальных магазинах. Каждая запись в файле содержит данные о названии инструмента, проданных единицах и стране продажи (разделенные символами табуляции):

Guitar 120 USA

Drums 80 Germany
Guitar 50 Canada
Piano 30 USA

Требуется вычислить среднее количество проданных единиц каждого музыкального инструмента в стране *USA*.

4. Подсчитать среднюю длину слов в нескольких файлах. Результат должен содержать средний размер слов в файле и соответствующее название файла. Сохранить результат в файл в виде: *(7@file1 6@file1 13@file2 22@file2 ...*).

5. Представим, что у нас есть файл с данными о продажах различных видов спортивных товаров в разных спортивных магазинах. Каждая строка файла содержит информацию о наименовании товара, проданных единицах и типе спортивного магазина (разделенные запятыми):

SoccerBall, 200, SportsStore
Basketball, 150, SportsStore
RunningShoes, 120, ShoeShop
SoccerBall, 100, ShoeShop

Нужно определить общее количество проданных единиц *SoccerBall* в магазинах типа *SportsStore*.

6. Предположим, у нас есть набор файлов с текстовым содержимым на различные темы. Каждый файл содержит слова, разделенные пробелами. Например, в файле *fruits* записано: *apple banana orange apple banana*. Нужно определить общее количество уникальных слов в каждом документе и вывести результат в формате "*Количество@Название файла*".

7. Для каждого пользователя подсчитать среднюю оценку, которую он поставил всем фильмам. Входной 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*.

8. Допустим, у нас есть файл с данными о продажах разных видов электроники в разных городах. Каждая запись в файле содержит

информацию о наименовании товара, проданных штуках и регионе продажи (разделены пробелами):

Laptop 50 City-X
Smartphone 30 City-Y
Laptop 20 City-Y
Tablet 40 City-X

Требуется вычислить среднее число проданных штук каждого товара в *City-X*.

9. Подсчитать среднюю стоимость показа рекламы по городам России и вывести максимальную стоимость. Входной файл имеет формат: *userId, country, city, campaign_id, creative_id, payment*. Результат должен быть сохранен в двух файлах:

1 файл: *city, av_av_payment*
2 файл (1 запись): *max_payment*

10. Предположим, у нас есть файл с данными о продажах разных видов кондитерских изделий в различных кафе города. Каждая строка файла содержит информацию о названии изделия, количестве проданных порций и районе города (разделенные точкой с запятой):

Cake; 80; Downtown
Cookies; 120; Suburb
Cupcakes; 50; Downtown
Cookies; 60; Suburb

Необходимо определить общее количество проданных порций каждого вида кондитерского изделия в районе *Downtown*.

11. Вывести в файл список фильмов относящиеся к заданному жанру в отсортированном виде (1-с наилучшим рейтингом). Входной файл имеет формат: *movieId, movieGenre, avgRating*. Результат должен быть сохранен в файле в формате: *moiveId, avg_rating*.

12. У нас есть набор файлов с логами веб-сервера. Каждая запись содержит информацию о запросе, времени и IP-адресе пользователя, разделенные табуляцией:

GET /page1 2023-01-15 192.168.1.1
POST /page2 2023-01-15 192.168.1.2
GET /page1 2023-01-15 192.168.1.1

Требуется определить общее количество запросов от каждого уникального IP-адреса.

13. В файле содержатся записи о продажах различных продуктов в разных магазинах с указанием названия продукта, проданных единиц и города продаж (разделенные запятыми) в следующем формате:

ProductA, 120, City-1

ProductB, 90, City-2

ProductA, 200, City-3

ProductC, 150, City-1

Требуется определить среднее количество проданных единиц каждого продукта в *City-1*.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Опишите концепцию цепочки MapReduce задач.
2. Перечислите стадии, из которых состоит MapReduce.
3. Приведите наиболее вычислительно трудную стадию MapReduce.
4. Раскройте значение термина Map-Only задачи.
5. Дайте определение понятию ChainMapper.
6. Дайте определение понятию ChainReducer.
7. Дайте определение комбинирующей функции.
8. Приведите примеры задач, для которых применение комбинирующих функций поможет повысить производительность.
9. Приведите примеры задач, для которых неприменимы комбинирующие функции.
10. Приведите пример задач, которые невозможно решить одной MapReduce стадией.
11. Раскройте область применения цепочек MapReduce задач.
12. 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 c. — 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 c. — 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 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/69375.html

Электронные ресурсы:

10. https://pig.apache.org/ (англ.)
11. https://www.tutorialspoint.com/apache_pig/index.htm (англ.)
12. http://hadoop.apache.org/ (англ.)