МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

**Инженерно-технологическая академия**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра Математического обеспечения и применения ЭВМ**

**ОТЧЁТ**

по лабораторной работе 1

по курсу «РиОВПП»

Выполнили:

ст. группы КТмо1-3

Шепель И. О.

Куприянова А. А.

Проверил:

преподаватель МОП ЭВМ

Пирская Л. В.

Оценка

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2017 г.

Таганрог 2017

**Цель работы:**

Реализовать алгоритм MapReduce в системе Apache Hadoop применительно к следующим трем задачам:

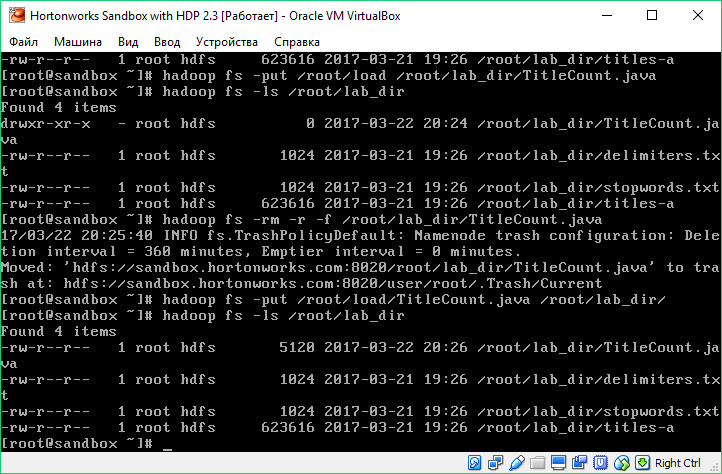
1. Подсчет количество слов в файле, с предварительным разбиением фраз на слова и удалением стоп-слов;

2. То же, что и в задании 1, затем организовать сортировку и вывести список из N наиболее частых слов. N задается как параметр.

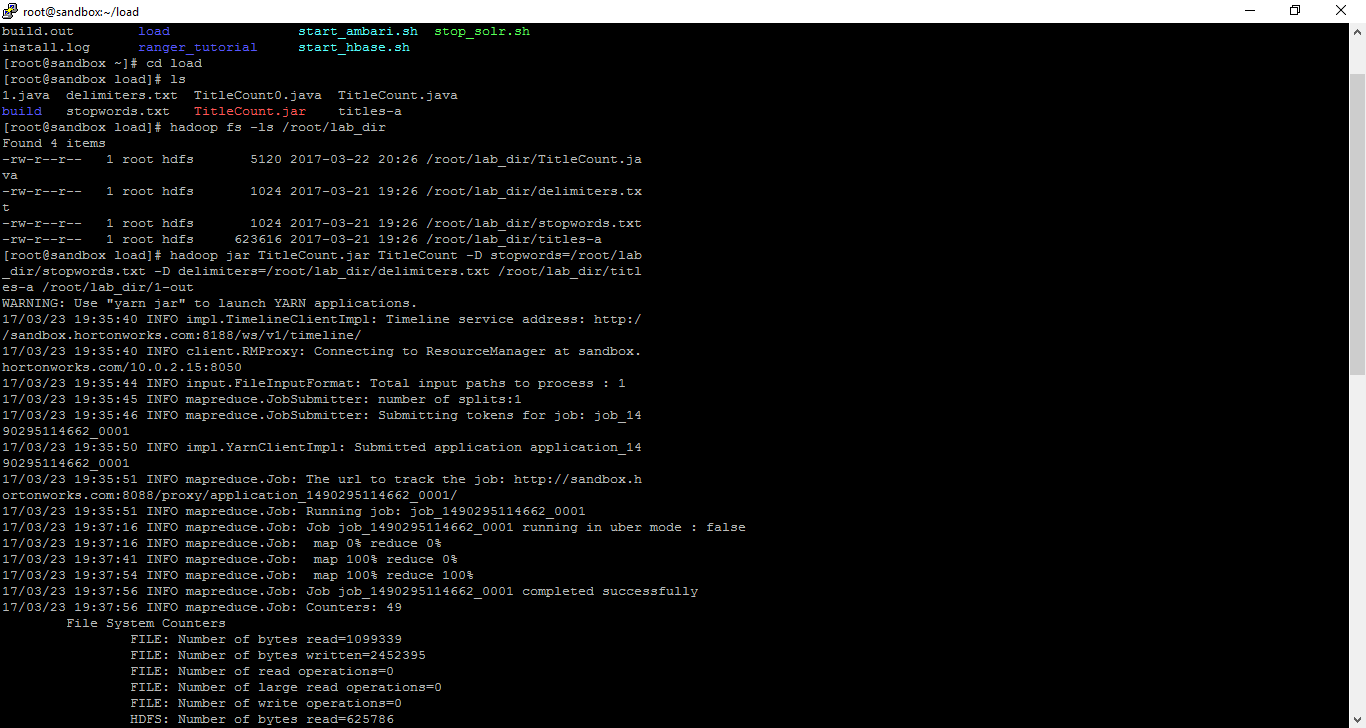
3. То же, что и в задании 2, затем организовать подсчет статистики по найденным словам, а именно – наименьшее и наибольшее значение, среднее, сумма и дисперсия выборки.

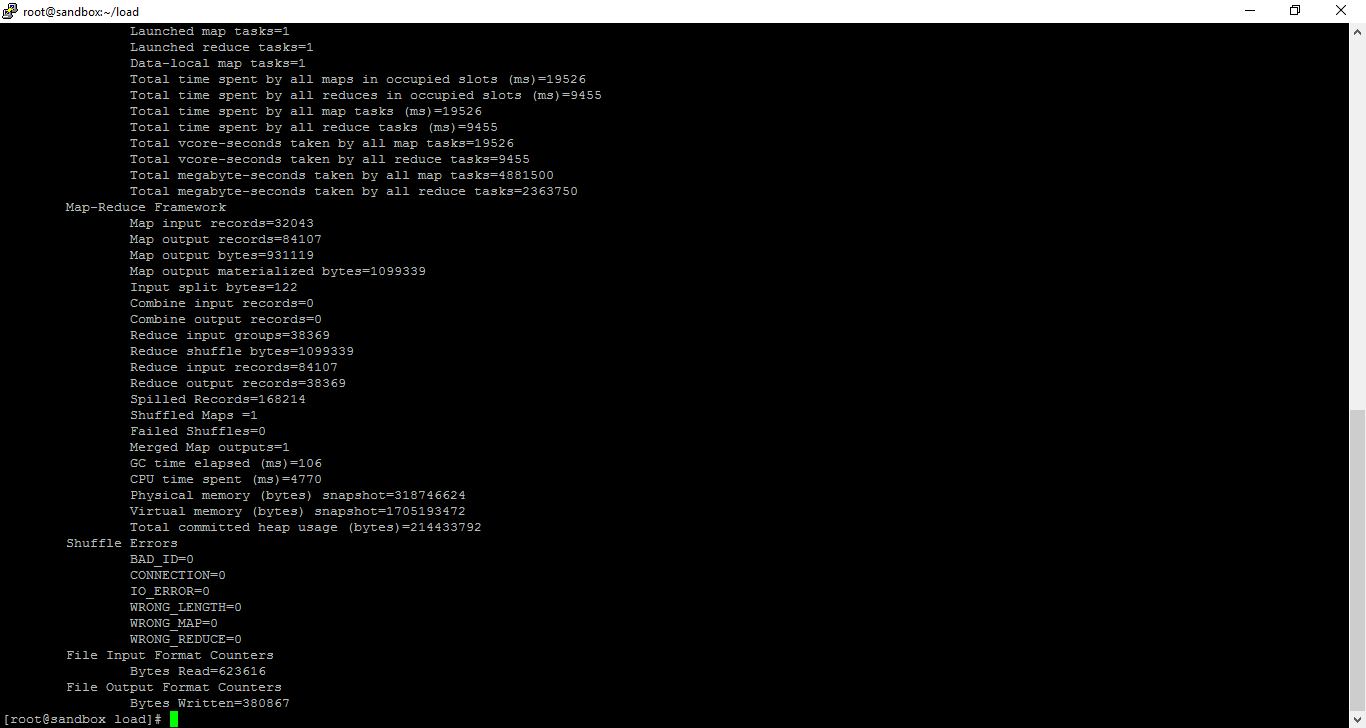
Процесс работы в системе Hadoop организован следующим образом:

Сперва файлы с исходными данными через консоль Ambari помещаются в локальную файловую систему. Затем оттуда мы перемещаем их в ранее созданную систему файлов HDFS.



Далее создаем файл скрипта, компилируем его и запускаем.

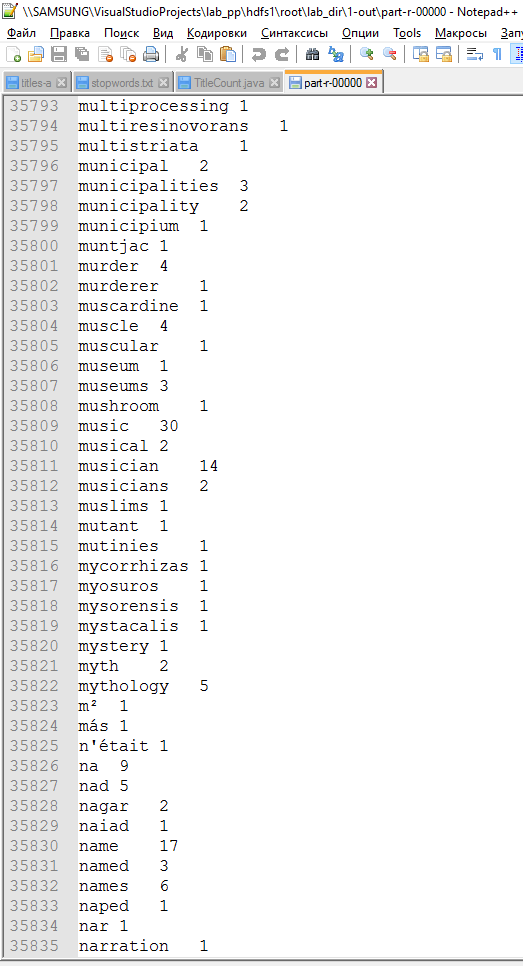




Приведем код 1-ого файла TitleCount.java, отвечающий за Map-Reduce:

**public static class** TitleCountMap **extends** Mapper<Object, Text, Text, IntWritable> {  
 List<String> stopWords;  
 String delimiters;  
  
 @Override  
 **protected void** setup(Context context) **throws** IOException,InterruptedException {  
  
 Configuration conf = context.getConfiguration();  
  
 String stopWordsPath = conf.get(**"stopwords"**);  
 String delimitersPath = conf.get(**"delimiters"**);  
  
 **this**.stopWords = Arrays.asList(readHDFSFile(stopWordsPath, conf).split(**"\n"**));  
 **this**.delimiters = readHDFSFile(delimitersPath, conf);  
 }  
  
 @Override  
 **public void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String line = value.toString();  
 StringTokenizer tokenizer = **new** StringTokenizer(line, **this**.delimiters);  
 **while** (tokenizer.hasMoreTokens()) {  
 String nextToken = tokenizer.nextToken();  
 nextToken = nextToken.toLowerCase();  
 **if** (!**this**.stopWords.contains(nextToken)){  
 context.write(**new** Text(nextToken), **new** IntWritable(1));  
 }  
 }  
 }  
}  
  
**public static class** TitleCountReduce **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
 @Override  
 **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));  
 }  
}

В методе setup map класса мы считываем наши параметры – слова разделители и стоп-слова, а в методе map – производим разбиение на токены, и затем составляем словарь, где каждому слову ставим в соответствие 1. Этот словарь и передается в reducer, который затем выполняет простое суммирование.

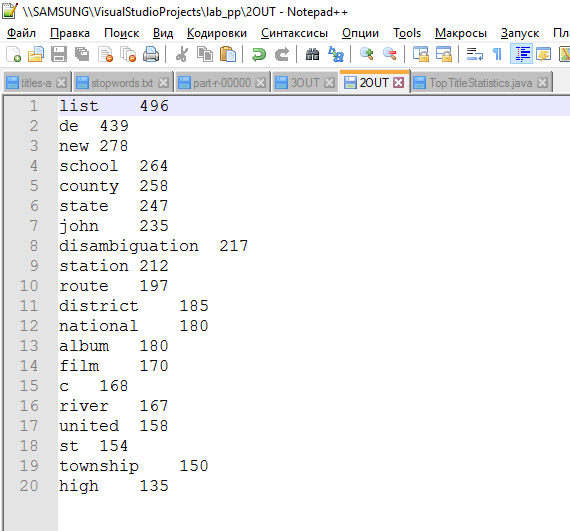
Скриншот с частичным результатом второго файла.  


Приведем код 2-ого файла TopTitles.java, отвечающий за Map-Reduce:

**public static class** TitleCountMap **extends** Mapper<Object, Text, Text, IntWritable> {  
 List<String> stopWords;  
 String delimiters;  
  
 @Override  
 **protected void** setup(Context context) **throws** IOException,InterruptedException {  
  
 Configuration conf = context.getConfiguration();  
  
 String stopWordsPath = conf.get(**"stopwords"**);  
 String delimitersPath = conf.get(**"delimiters"**);  
  
 **this**.stopWords = Arrays.asList(readHDFSFile(stopWordsPath, conf).split(**"\n"**));  
 **this**.delimiters = readHDFSFile(delimitersPath, conf);  
 }  
  
  
 @Override  
 **public void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
String line = value.toString();  
 StringTokenizer tokenizer = **new** StringTokenizer(line, **this**.delimiters);  
 **while** (tokenizer.hasMoreTokens()) {  
 String nextToken = tokenizer.nextToken();  
 nextToken = nextToken.toLowerCase();  
 **if** (!**this**.stopWords.contains(nextToken)){  
 context.write(**new** Text(nextToken), **new** IntWritable(1));  
 }  
 }  
 }  
 }  
  
 **public static class** TitleCountReduce **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
 @Override  
 **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 class** TopTitlesMap **extends** Mapper<Text, Text, NullWritable, TextArrayWritable> {  
 Integer N;  
 SortedSet sortedSet;  
  
 @Override  
 **protected void** setup(Context context) **throws** IOException,InterruptedException {  
 Configuration conf = context.getConfiguration();  
 **this**.N = conf.getInt(**"N"**, 10);  
  
 **this**.sortedSet = **new** TreeSet();  
 }  
  
 @Override  
 **public void** map(Text key, Text value, Context context) **throws** IOException, InterruptedException {  
 String keyS = key.toString();  
Integer valS = Integer.valueOf(value.toString());  
**this**.sortedSet.add(**new** Pair<Integer, String>(valS, keyS));  
 }  
  
 @Override  
 **protected void** cleanup(Context context) **throws** IOException, InterruptedException {  
 String[] strings = **new** String[**this**.N];  
 **for** (**int** i = 0; i < N; i ++){  
 Pair<Integer, String> pair = (Pair<Integer, String>) **this**.sortedSet.last();  
 strings[i] = pair.toString();  
 sortedSet.remove(pair);  
 }  
 context.write(NullWritable.get(), **new** TextArrayWritable(strings));  
 }  
 }  
  
 **public static class** TopTitlesReduce **extends** Reducer<NullWritable, TextArrayWritable, Text, IntWritable> {  
 Integer N;  
 *// TODO* @Override  
 **protected void** setup(Context context) **throws** IOException,InterruptedException {  
 Configuration conf = context.getConfiguration();  
 **this**.N = conf.getInt(**"N"**, 10);  
 }  
  
 @Override  
 **public void** reduce(NullWritable key, Iterable<TextArrayWritable> values, Context context) **throws** IOException, InterruptedException {  
 **for** (TextArrayWritable textArrayWritable : values){  
 **for** (String s : textArrayWritable.toStrings()){  
 String[] parts = s.split(**", "**);  
 context.write(**new** Text(parts[1].substring(0, parts[1].length() - 1)),  
 **new** IntWritable(Integer.parseInt(parts[0].substring(1, parts[0].length()))));  
 }  
 }  
 }  
 }

Здесь Map-Reduce происходит в два этапа, и первый из них идентичен первому заданию. Рассмотрим второй. На входе маппера мы пакуем все наши записи в структуру SortedSet, которая автоматически проводит сортировку по первому значению – количеству повторений слова. Затем в методе cleanup, который выполняется 1 раз после всех выполнений метода map (для каждой строки – по 1му) [1], мы передаем только N наибольших по количеству записей. Затем в reducer-е мы просто переворачиваем нашу строку из вида число-слово в вид слово-число.

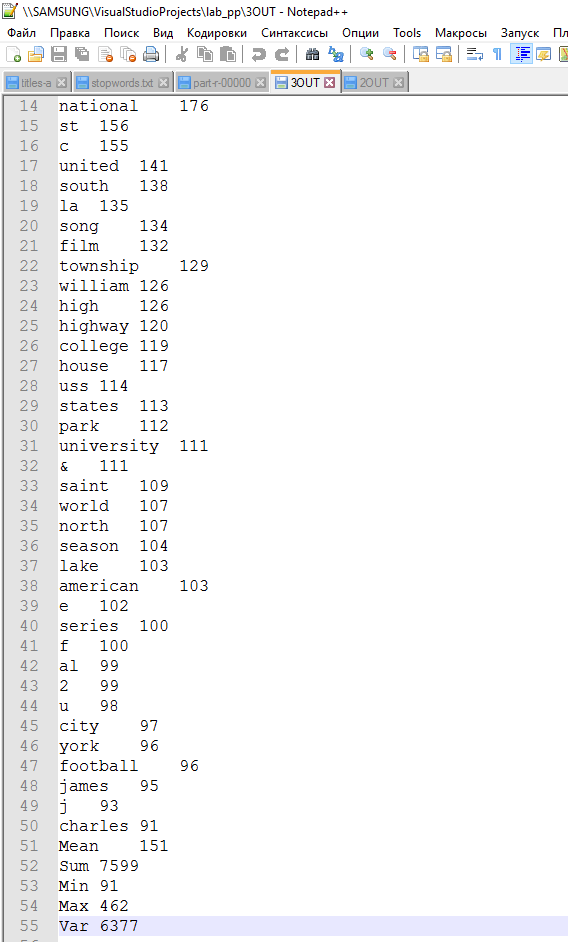
Результат - скриншот второго файла для N=20:



Приведем код 3-ого файла TopTitleStatistics.java, отвечающий за Map-Reduce:

**public static class** TitleCountMap **extends** Mapper<Object, Text, Text, IntWritable> {  
 List<String> stopWords;  
 String delimiters;  
  
 @Override  
 **protected void** setup(Context context) **throws** IOException,InterruptedException {  
  
 Configuration conf = context.getConfiguration();  
  
 String stopWordsPath = conf.get(**"stopwords"**);  
 String delimitersPath = conf.get(**"delimiters"**);  
  
 **this**.stopWords = Arrays.asList(readHDFSFile(stopWordsPath, conf).split(**"\n"**));  
 **this**.delimiters = readHDFSFile(delimitersPath, conf);  
 }  
  
  
 @Override  
 **public void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  
 String line = value.toString();  
 StringTokenizer tokenizer = **new** StringTokenizer(line, **this**.delimiters);  
 **while** (tokenizer.hasMoreTokens()) {  
 String nextToken = tokenizer.nextToken();  
 nextToken = nextToken.toLowerCase();  
 **if** (!**this**.stopWords.contains(nextToken)){  
 context.write(**new** Text(nextToken), **new** IntWritable(1));  
 }  
 }  
 }  
 }  
  
 **public static class** TitleCountReduce **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
 @Override  
 **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 class** TopTitlesStatMap **extends** Mapper<Text, Text, NullWritable, TextArrayWritable> {  
 Integer N;  
 SortedSet sortedSet;  
  
 @Override  
 **protected void** setup(Context context) **throws** IOException,InterruptedException {  
 Configuration conf = context.getConfiguration();  
 **this**.N = conf.getInt(**"N"**, 10);  
 **this**.sortedSet = **new** TreeSet();  
 }  
  
 @Override  
 **public void** map(Text key, Text value, Context context) **throws** IOException, InterruptedException {  
 String keyS = key.toString();  
 Integer valS = Integer.valueOf(value.toString());  
 **this**.sortedSet.add(**new** Pair<Integer, String>(valS, keyS));  
 }  
  
 @Override  
 **protected void** cleanup(Context context) **throws** IOException, InterruptedException {  
 String[] strings = **new** String[**this**.N];  
 **for** (**int** i = 0; i < N; i ++){  
 Pair<Integer, String> pair = (Pair<Integer, String>) **this**.sortedSet.last();  
 strings[i] = pair.toString();  
 sortedSet.remove(pair);  
 }  
 context.write(NullWritable.get(), **new** TextArrayWritable(strings));  
 }  
 }  
  
 **public static class** TopTitlesStatReduce **extends** Reducer<NullWritable, TextArrayWritable, Text, IntWritable> {  
 Integer N;  
 *// TODO* @Override  
 **protected void** setup(Context context) **throws** IOException,InterruptedException {  
 Configuration conf = context.getConfiguration();  
 **this**.N = conf.getInt(**"N"**, 10);  
 }  
  
 @Override  
 **public void** reduce(NullWritable key, Iterable<TextArrayWritable> values, Context context) **throws** IOException, InterruptedException {  
 Integer sum, mean, max, min, var, sq\_sum;  
 sum = 0;  
 sq\_sum = 0;  
 **int** i = 0;  
 **for** (TextArrayWritable textArrayWritable : values){  
 **for** (String s : textArrayWritable.toStrings()){  
 String[] parts = s.split(**", "**);  
 Integer val = Integer.parseInt(parts[0].substring(1, parts[0].length()));  
 **if** (i==0){  
 max = val;  
 min = val;  
 }  
 **else** {  
 **if** (max < val){  
 max = val;  
 }  
 **if** (min > val){  
 min = val;  
 }  
 }  
 sum = sum + val;  
 sq\_sum = sq\_sum + val \* val;  
 context.write(**new** Text(parts[1].substring(0, parts[1].length() - 1)),  
 **new** IntWritable(Integer.parseInt(parts[0].substring(1, parts[0].length()))));  
 i = i+1;  
 }  
 }  
  
 mean = (Integer) sum / **this**.N;  
 var = (Integer) (sq\_sum / **this**.N) - mean \* mean;  
  
 context.write(**new** Text(**"Mean"**), **new** IntWritable(mean));  
 context.write(**new** Text(**"Sum"**), **new** IntWritable(sum));  
 context.write(**new** Text(**"Min"**), **new** IntWritable(min));  
 context.write(**new** Text(**"Max"**), **new** IntWritable(max));  
 context.write(**new** Text(**"Var"**), **new** IntWritable(var));  
 }  
 }  
}

Принцип работы и код полностью аналогичны второму примеру, только в методе reduce теперь производится и подсчет статистики по общеизвестным формулам мат. статистики.

Результат: 

**Список использованных источников.**

1. Документация Apache Hadoop <https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Mapper.html>