ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КУРСОВАЯ РАБОТА

по дисциплине “Распределенная обработка информации”

на тему

MapReduce-алгоритм построения инвертированного индекса

Выполнил студент Макаревич А.А.
Ф.И.О.

Группы ЗМП-01

Работу принял _____ профессор д.т.н. М.Г. Курносов
подпись

Защищена _____ Оценка _____

Новосибирск – 2021

СОДЕРЖАНИЕ

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Требуется построить инвертированный индекс (inverted index) для заданного корпусатекстов (текстового файла).

Входные данные map:

(docid,
content)

Результирующий инвертированный индекс должен иметь следующую структуру:

(word, [<docid1, TF-IDF1>, <docid2, TF-IDF2>,
...])

- Статьи должны быть отсортированы в порядке убывания TF-IDF(Term Frequency – Inverse Document Frequency)
- Для каждого слова ограничить список статей *N* наиболее релевантными
- Определить и исключить из индекса Top20 высокочастотных слов

При вычислении TF-IDF считаем, что:

- *TF*(*t*, *d*) — это число вхождений слова *t* в документ *d* (Wiki-статью)
- *IDF*(*t*, *D*) — обратная частота, с которой слово *t* встречается во множестведокументов *D* (Wiki-статьях):

$$\mathrm{IDF}(t,D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

Программы должны быть написаны на языке Java (Apache Hadoop Java API)

ВВЕДЕНИЕ

С появлением вычислительной техники объем информации, с которой может работать человек, значительно увеличился. Еще более 30 лет назад учеными стали разрабатываться алгоритмы, позволяющие упростить работу с данными, а также получить из них новые, ранее неизвестные знания. Однако сегодня эти алгоритмы неэффективны, поскольку объемы информации стали слишком большими. Для работы с большими объемами данных была разработана специальная система Apache Hadoop, Главное место в этой системе занимает технология под названием MapReduce. Реализация MapReduce позволяет производить распределенные вычисления над большими объемами данных в компьютерных кластерах эффективно и безотказно.

К тому же, технология MapReduce быстро развивается и распространяется. Многие компании используют ее для обработки данных, потому что она имеет открытый исходный код, масштабируема и не требует больших затрат на оборудование. Новизна исследования состоит в том, что многие проблемы, возникающие, при обработке больших данных могут быть решены за счет высокой отказоустойчивости, масштабируемости и доступности технологии MapReduce.

I. Инвертированный индекс в MapReduce

В существующих механизмах веб-поиска используются распределенные алгоритмы индексирования, которые строят распределенные индексы. Индекс разделяется на несколько компьютеров в соответствии с термином, либо с документом. Такой алгоритм строится на модели распределенных вычислений MapReduce. MapReduce работает на больших компьютерных кластерах, где каждый узел является 11 обычным компьютером. Эти узлы управляются главным узлом, который назначает задачи рабочим узлам. MapReduce имеет две фазы:

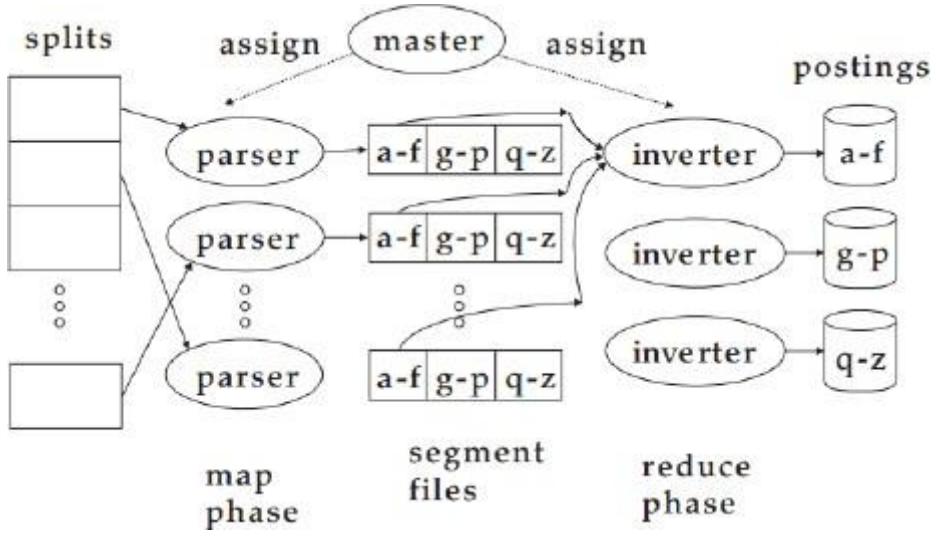

Рисунок 1. – Инвертированный индекс в MapReduce.

Map. На этом этапе главный узел сначала разбивает коллекцию документов на одинаковые по размеру части, которые в дальнейшем будут равномерно распределены между узлами. Затем главный узел назначает элементы рабочим узлам, которые называются парсерами или синтаксическими анализаторами. Процесс синтаксического анализа очень похож на процесс, выполняемый в нераспределенных алгоритмах. После обработки каждый парсер записывает результат (пары ключ-значение) на локальный диск рабочего узла.

Reduce. На данном этапе происходит сортировка и агрегация пар ключ-значение по ключу и передача их на инверторный рабочий узел. Инверторы собирают и сортируют все docID для данного termID, и в результате получают данные в виде <ключ, список документов>.

Был рассмотрен ряд алгоритмов построения инвертированного индекса. Для реализации был выбран распределенный алгоритм, построенный на модели

MapReduce, так как он единственный из рассмотренных алгоритмов способен работать с большими коллекциями данных.

Для реализации алгоритма построения инвертированного индекса, основанного на модели MapReduce, необходимо детально разобрать принцип работы этой парадигмы.

II. Парадигма MapReduce

MapReduce - это модель распределенных вычислений, представленная компанией Google в 2004 году для сканирования и обработки множества страниц из сети Интернет. Первая реализация этой модели была выполнена на основе распределенной файловой системы GFS (Google File System). Эта реализация запатентована и активно используется компанией, но использование ее вне Google недоступно.

Альтернативная, свободно доступная реализация Hadoop MapReduce была выполнена в проекте Hadoop сообщества Apache. Эта реализация основывается на использовании распределенной файловой системы HDFS (Hadoop Distributed File System), также разработанной в проекте Hadoop. Реальную популярность MapReduce принесла именно реализация Hadoop в силу своей доступности и открытости, а широкое использование Hadoop MapReduce в различных исследовательских проектах приносит несомненную пользу этой системе, стимулируя разработчиков к ее постоянному совершенствованию.

Реализация MapReduce способна выполнять множество распределенных вычислений на сотнях или даже тысячах связанных между собой компьютеров таким образом, что сложность распараллеливания, обработка ошибок и аппаратные неисправности не являются проблемой. Все, что нужно разработчику - это написать две функции, которые называются Map и Reduce, в то время как система самостоятельно управляет распределенными вычислениями, координацией выполняющихся задач, а также контролирует ситуацию, если одна из этих задач не выполняется. Схема работы MapReduce изображена на рис. 2.

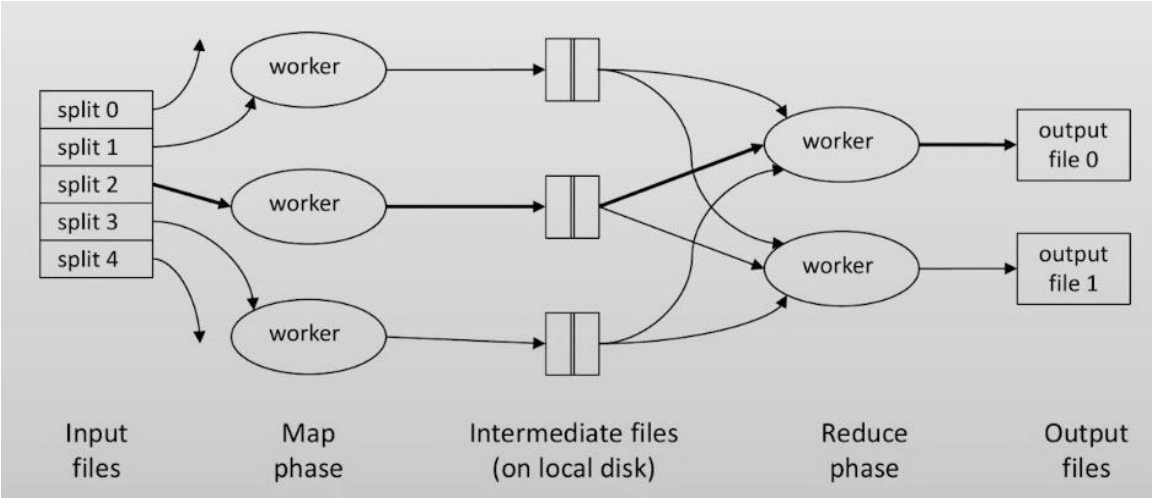

Рисунок 2. – Схема работы MapReduce.

Алгоритм выполнения MapReduce:

1.Часть данных поступает на один из вычислительных узлов и обрабатывается функцией Map. Выводом этой программы с одного кластера являются пары ключ-значение.

2. Все пары ключ-значение сортируются по ключу и передаются функции Reduce.

3. Функция Reduce обрабатывает данные, упорядоченные по ключу. Способ обработки определяет код, написанный пользователем для функции Reduce.

Это факт, что рано или поздно аппаратные средства выходят из строя, и чем больше вычислительных узлов и соединений имеет система, тем больше вероятность того, что произойдет сбой и остановка работы системы.

Некоторые вычисления могут занять несколько минут или даже часов в тысячах вычислительных узлов. Если бы приходилось прерывать и перезапускать вычисления каждый раз, когда один компонент выходит из строя, то система бы работала не эффективно.

В MapReduce эта проблема решается следующими способами:

1. Файлы имеют резервные копии. Если не дублировать файл на нескольких вычислительных узлах, то при выходе из строя узла, все его файлы станут недоступны, до тех пор, пока он не будет отремонтирован. А если произойдет сбой диска, то файлы будут потеряны навсегда.

2. Вычисления разделены на задачи таким образом, что, если одна задача не выполняется до конца, то она перезапускается, не затрагивая другие задачи.

Для того, чтобы использовать кластерные вычисления, файлы должны выглядеть несколько иначе, чем в обычных файловых системах. Для этого используется так называемая распределенная файловая система (Distributed File System). Она функционирует следующим образом:

Файлы разделены на части, как правило, размером по 128 МБ. Эти части реплицируются три раза в три различных вычислительных узла. Кроме того, каждая часть должна быть расположена на разных стойках, чтобы не произошло потери всех копий из-за выхода из строя всей стойки. Как правило, такие параметры, как размер блока данных и степень репликации задаются пользователем. Для того, чтобы найти части файла, существует специальный небольшой метафайл, которым управляет главный узел.

Входные данные для задачи Map состоят из независимых между собой элементов, которые могут быть любого вида: массив, документ, и тому подобное. Часть данных может представлять собой совокупность элементов. Все выходые данные функции Map и выходные данные функции Reduce представлены в виде пар ключ-значение. Функция Map принимает входной элемент в качестве аргумента и производит ноль или более пар ключ-значение. Типы ключей и значений каждый узел генерирует произвольно. Кроме того, ключи не должны быть уникальными. Чаще всего одна задача функции Map может произвести несколько пар ключ-значение с одинаковым ключом.

Как только задача Map успешно завершится, пары ключ-значение группируются по ключу, а значения каждого ключа формируются в список значений. Группировка осуществляется внутри системы, и не задается разработчиком. Разработчик обычно указывает системе количество Reduce задач (обозначим r). Во время выполнения главный контроллер выбирает хэш-функцию, которая применяется к ключам и производит ряд значений от 0 до r - 1. Каждый ключ из задачи Map хешируется и его пары ключ-значение записываются в один из r локальных файлов. Каждый файл предназначен для одной из Reduce задач. Чтобы выполнить группировку по ключу и распределить файлы по Reduce задачам, главный контроллер объединяет файлы,

предназначенные для конкретной Reduce задачи и передает объединенный файл в этот процесс в виде последовательности пар ключ-список значений.

Аргументом функции Reduce является ключ и список, соответствующих ему значений. Выходом функции Reduce является последовательность из нуля или более пар ключ-значение. Эти пары могут быть иного типа, нежели при выходе из Map задачи, но обычно их типы совпадают. Задача Reduce получает один или несколько ключей и связанных с ними списков значений. То есть, Reduce выполняет одно или несколько заданий. Выходы из всех Reduce задач объединяются в один файл. Иногда функция Reduce обладает свойствами ассоциативности и коммутативности. То есть, значения могут быть объединены в любом порядке, а результат останется неизменным.

На рисунке 3 изображена подробная схема, отображающая взаимодействие между файлами, процессами и задачами во время выполнения MapReduce программы. При помощи библиотек, представленных системой Hadoop MapReduce, главный узел производит разветвление процессов на различных вычислительных узлах. Как правило, на узлах обрабатывается задача Map, или Reduce, но не обе одновременно.

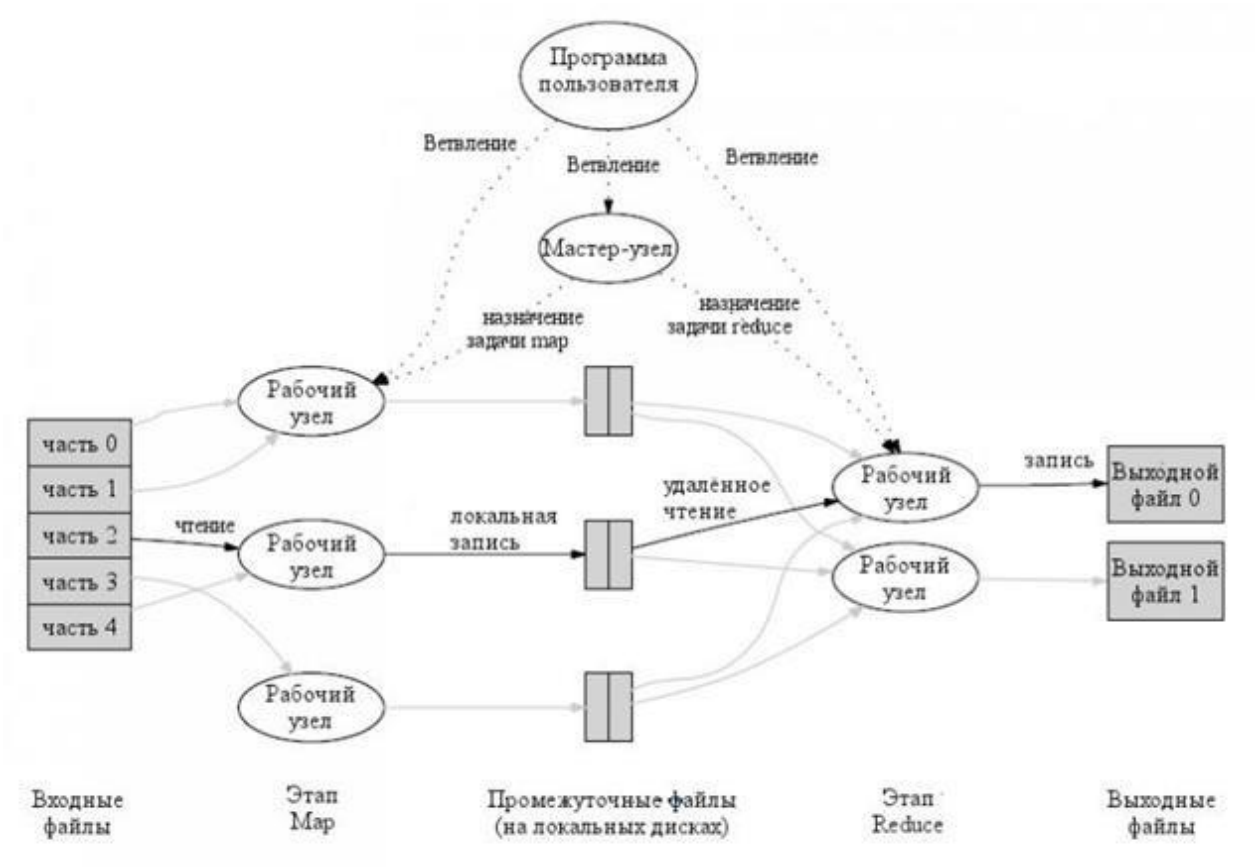

Рисунок 3. - Детальный обзор работы MapReduce программы.

Главный узел имеет много обязанностей. Одной из них является создание определенного количества Map и Reduce задач. Их количество задается разработчиком в программе. Эти задачи будут назначены на рабочие процессы с помощью главного узла. Разумно создавать по одной Map задачи для каждого фрагмента входного файла, тогда потребуется меньше Reduce задач. Причиной ограничения является необходимость создания промежуточных файлов для каждой Map задачи. Если Map задач будет слишком много, то число промежуточных файлов будет значительно больше числа Reduce задач. Поэтому перед тем, как задавать количество Map задач необходимо проанализировать возможности Reduce задач.

Главный процесс отслеживает состояние каждой Map и Reduce задачи. Когда рабочий процесс завершает выполнение задачи, главный процесс назначает ему новую.

Каждой Map задаче задается одна или несколько частей входного файла и после этого она выполняет код, написанный пользователем. Map задача создает

файл для каждой Reduce задачи на локальном диске узла, который выполняет задание. Главному процессу сообщается о месте и размере каждого из этих файлов. Когда Reduce задача назначается мастером в рабочий процесс, то ей передаются все необходимые файлы, формулирующие ее вход. Reduce задача выполняет код, написанный разработчиком, и записывает свой результат в файл, который является частью распределенной файловой системы.

Худшее, что может произойти - вычислительный узел, на котором работает главный процесс, перестанет работать. В этом случае вся MapReduce программа должна быть перезапущена. Это единственный узел, который может нанести ущерб всему процессу; другие отказы будут управляться главным узлом, и работа MapReduce будет завершена без больших задержек.

РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

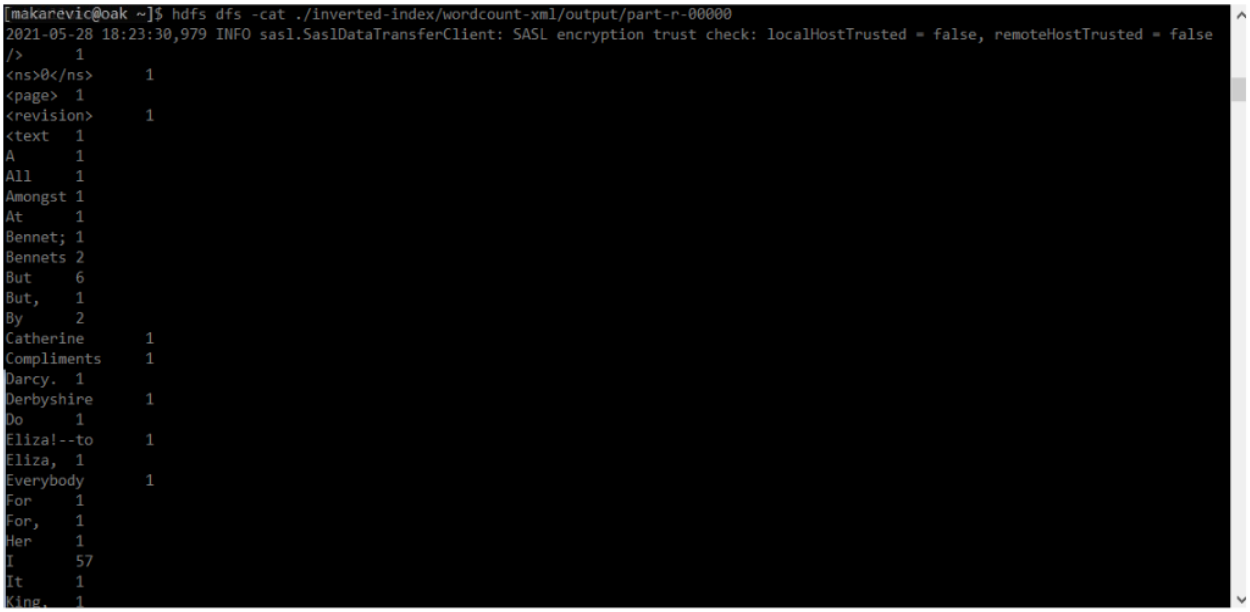

Рисунок 4. –Word Count.

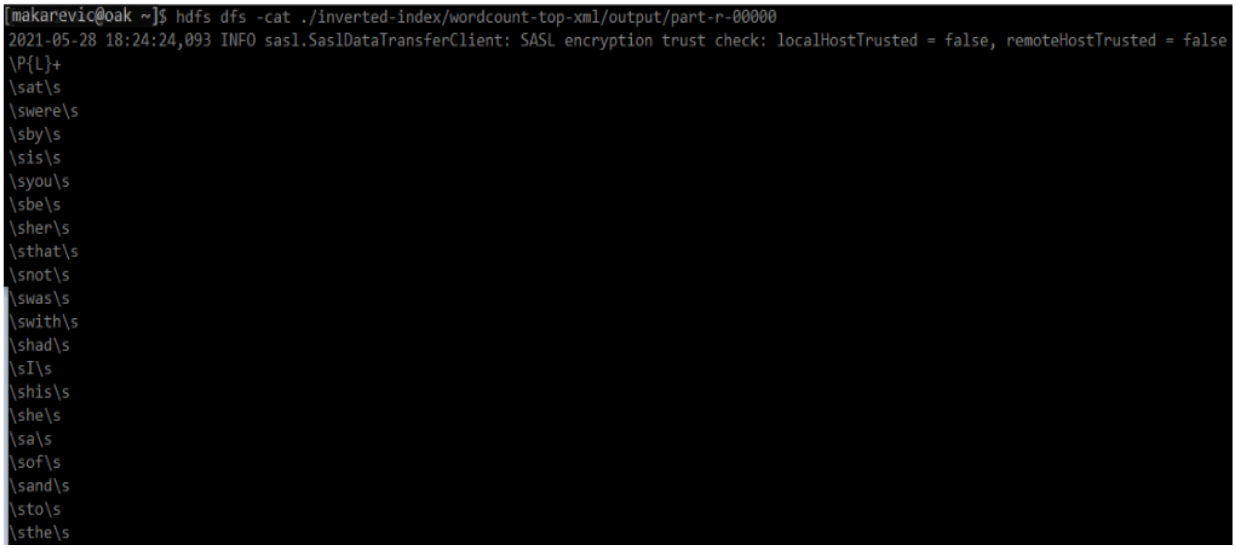

Рисунок 5. – Wordcount top.

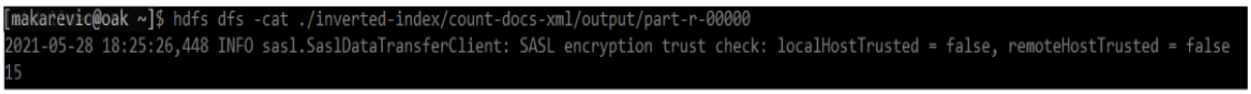

Рисунок 6. – Count docs.

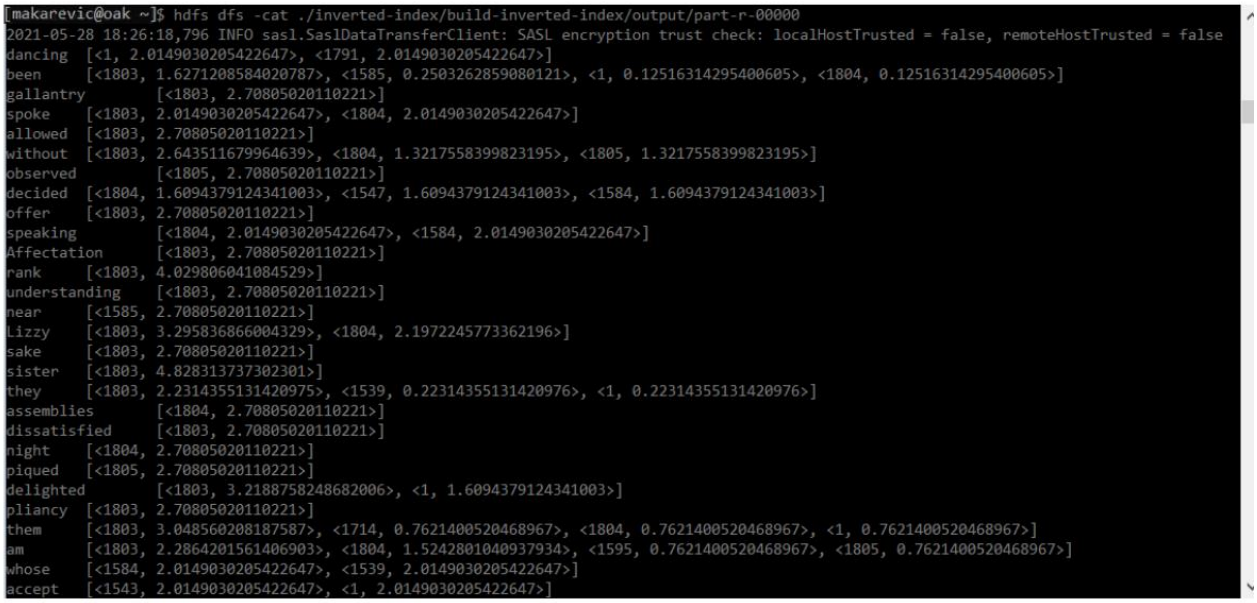

Рисунок 7. – Build inverted index.

ЗАКЛЮЧЕНИЕ

В курсовой работе рассмотрена технология MapReduce её надёжность, выяснили принципы работы технологии и её реализации.

В результате выполнения реализован MapReduce-алгоритм построения инвертированного индекса для заданного корпуса текстов.

На сегодняшний день уже понятно, что технология MapReduce может эффективно применяться внутри параллельной аналитической СУБД, служить инфраструктурой отказоустойчивой параллельной СУБД, а также сохранять свою автономность в симбиотическом союзе с параллельной СУБД. Все это не только не мешает развитию технологии параллельных СУБД, а наоборот, способствует ее совершенствованию и распространению.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хорошевский В.Г. Архитектура вычислительных систем. – М.: МГТУ им. Н.Э. Баумана, 2008. – 520 с.
2. Евреинов Э.В., Хорошевский В.Г. Однородные вычислительные системы. – Новосибирск: Наука, 1978. – 320 с.
3. Rabenseifner R.. Automatic MPI Counter Profiling // Proceedings of the 42nd Cray User Group. – Noorwijk, The Netherlands, 2000. – 19 pp.
4. Han D., Jones T.. MPI Profiling // Technical Report UCRL-MI-209658 – Lawrence Livermore National Laboratory, USA, 2004. – 15 pp.
5. Thakur R., Rabenseifner R., and Gropp W. Optimization of collective communication operations in MPICH // Int. Journal of High Performance Computing Applications. – 2005. – Vol. 19, No. 1. – P. 49-66.
6. Balaji P., Buntinas D., Goodell D., Gropp W., Kumar S., Lusk E., Thakur R. and Traff J. L. MPI on a Million Processors // Proc. of the PVM/MPI – Berlin: Springer-Verlag, 2009. – P. 20-30.
7. Khoroshevsky V., Kurnosov M. Mapping Parallel Programs into Hierarchical Distributed Computer Systems // Proc. of “Software and Data Technologies”. – Sofia: INSTICC, 2009. – Vol. 2. – P. 123-128.

ПРИЛОЖЕНИЕ А

WordCount.java

```
package pdccourse.hw3;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Text, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Text 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();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", " ");
    conf.setBoolean("exact.match.only", true);
    conf.set("io.serializations",
          "org.apache.hadoop.io.serializer.JavaSerialization,"
          + "org.apache.hadoop.io.serializer.WritableSerialization");

    Job job = new Job(conf, "word count");
    job.setInputFormatClass(XmlInputFormat.class);
    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(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

WordCountTop.java

```
package pdccourse.hw3;

import java.io.IOException;
import java.util.StringTokenizer;

import java.util.Map;
import java.util.Map.Entry;
import java.util.TreeMap;

import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.conf.Configuration;
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.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
```

```
public class WordCountTop {

  public static class TokenizerMapper
       extends Mapper<Text, Text, Text, LongWritable> {
    private TreeMap<Long, String> tmap;

    @Override
    public void setup(Context context) throws IOException, InterruptedException {
      tmap = new TreeMap<Long, String>();
    }

    public void map(Text key, Text value, Context context) throws IOException,
InterruptedException {
      String word = key.toString();
      Long count = Long.parseLong(value.toString());

      tmap.put(count, word);
      if (tmap.size() > 20) {
        tmap.remove(tmap.firstKey());
      }
    }

    @Override
    public void cleanup(Context context) throws IOException, InterruptedException {
      for (Map.Entry<Long, String> entry : tmap.entrySet()) {
          context.write(new Text(entry.getValue()), new LongWritable(entry.getKey()));
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text, LongWritable, Text, LongWritable> {
    private TreeMap<Long, String> tmap;

    @Override
    public void setup(Context context) throws IOException, InterruptedException {
      tmap = new TreeMap<Long, String>();
    }

    public void reduce(Text key, Iterable<LongWritable> values, Context context
                ) throws IOException, InterruptedException {
      String word = key.toString();
      long count = 0;
      for (LongWritable val : values) {
        count = val.get();
      }

      tmap.put(count, word);
      if (tmap.size() > 20) {
          tmap.remove(tmap.firstKey());
      }
```

```
    }

    @Override
    public void cleanup(Context context) throws IOException, InterruptedException {
      context.write(new Text("\\P{L}+"), null);
      for (Map.Entry<Long, String> entry : tmap.entrySet()) {
          String txt = "\\s" + entry.getValue() + "\\s";
          //String txt = entry.getValue();
          context.write(new Text(txt), null);
      }
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: WordCountTop <in> <out>");
      System.exit(2);
    }
    conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", "\t");
    conf.setBoolean("exact.match.only", true);
    conf.set("io.serializations",
          "org.apache.hadoop.io.serializer.JavaSerialization,"
          + "org.apache.hadoop.io.serializer.WritableSerialization");

    Job job = new Job(conf, "word count top");
    job.setInputFormatClass(KeyValueTextInputFormat.class);
    job.setJarByClass(WordCountTop.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(LongWritable.class);
    //job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

CountDocs.java

```
package pdccourse.hw3;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
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.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class CountDocs {

  public static class TokenizerMapper
       extends Mapper<Text, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);

    public void map(Text key, Text value, Context context) throws IOException,
InterruptedException {
      context.write(key, one);
    }
  }

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

    static enum Counters {
      COUNT_DOCS
    }

    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.getCounter(Counters.COUNT_DOCS).increment(1);
      //context.write(key, result);
    }

    @Override
    public void cleanup(Context context) throws IOException, InterruptedException {
      context.write(new LongWritable(context.getCounter(Counters.COUNT_DOCS).getValue()),
null);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
```

```
    if (otherArgs.length != 2) {
      System.err.println("Usage: countdocs <in> <out>");
      System.exit(2);
    }
    //conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", " ");
    conf.setBoolean("exact.match.only", true);
    conf.set("io.serializations",
          "org.apache.hadoop.io.serializer.JavaSerialization,"
          + "org.apache.hadoop.io.serializer.WritableSerialization");

    Job job = new Job(conf, "count docs");
    job.setInputFormatClass(XmlInputFormat.class);
    job.setJarByClass(CountDocs.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    //job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(LongWritable.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

BuildInvertedIndex.java

```
package pdccourse.hw3;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.File;
import java.io.FileReader;

import java.util.StringTokenizer;
import java.util.Map;
import java.util.Map.Entry;
import java.util.HashMap;
import java.util.List;
import java.util.ArrayList;
import java.util.LinkedHashMap;

import org.apache.hadoop.conf.Configuration;
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.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
```

```
import org.apache.hadoop.io.WritableUtils;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.StringUtils;

public class BuildInvertedIndex {

  public static class MPair implements WritableComparable<MPair> {
    private String word;
    private Integer tf;

    public void set(String word, Integer tf) {
      this.word = word;
      this.tf = tf;
    }

    public String getWord() {
      return word;
    }

    public Integer getTf() {
      return tf;
    }

    @Override
    public void readFields(DataInput in) throws IOException {
      word = Text.readString(in);
      tf = Integer.parseInt(Text.readString(in));
    }

    @Override
    public void write(DataOutput out) throws IOException {
      Text.writeString(out, word);
      Text.writeString(out, tf.toString());
    }

    @Override
    public int hashCode() {
      return word.hashCode();
    }

    @Override
    public String toString() {
      return word.toString();
    }
```

```
    @Override
    public int compareTo(MPair o) {
      //String[] wr = word.split(" ");
      //String[] owr = o.word.split(" ");
      if (!word.equals(o.word)) {
         return word.compareTo(o.word);
      } else {
          //if (!wr[1].equals(owr[1])) {
            return (o.tf > tf ? 1 : -1);
          //}
          //return 0;
      }
    }
  }

  public static class MPartitioner extends Partitioner<MPair, Text> {
    @Override
    public int getPartition(MPair pair, Text docid, int numOfPartitions) {
      return ( pair.getWord().hashCode() & Integer.MAX_VALUE ) % numOfPartitions;
    }
  }

  public static class MGroupComparator extends WritableComparator {
    //private static final Text.Comparator TEXT_COMPARATOR = new Text.Comparator();
    //private static final IntWritable.Comparator INTWRITABLE_COMPARATOR = new
IntWritable.Comparator();

    public MGroupComparator() {
      super(MPair.class, true);
    }

    /*@Override
    public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
      try {
          int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1, s1);
          int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2);
          int cmp1 = TEXT_COMPARATOR.compare(b1, s1, firstL1, b2, s2, firstL2);
          if (cmp1 != 0) {
            return cmp1;
          } else {
            int secondL1 = WritableUtils.decodeVIntSize(b1[s1+firstL1]) + readVInt(b1, s1+firstL1);
            int secondL2 = WritableUtils.decodeVIntSize(b2[s2+firstL2]) + readVInt(b2, s2+firstL2);
            return (-1) * INTWRITABLE_COMPARATOR.compare(b1, s1+firstL1, secondL1, b2,
s2+firstL2, secondL2);
          }
      } catch (IOException e) {
          throw new IllegalArgumentException(e);
      }
    }*/

    @Override
    public int compare(WritableComparable w1, WritableComparable w2) {
```

```
    if (w1 instanceof MPair && w2 instanceof MPair) {
        return ((MPair)w1).compareTo((MPair)w2);
    }
    return super.compare(w1, w2);
  }
 }

 public static class TokenizerMapper
     extends Mapper<Text, Text, MPair, Text> {

  private List<String> skipList = new ArrayList<String>();
  private long numRecords = 0;
  private Map<String, Map<String, Integer> > results = new HashMap<String, Map<String,
Integer> >();
  private final MPair pair = new MPair();
  private final Text docid = new Text();

  @Override
  protected void setup(Context context) {
   String skipListFile = context.getConfiguration().get("buildinvertedindex.skip-list");
   if (skipListFile != null) {
       loadSkipListFile(skipListFile);
   }
  }

  public void map(Text key, Text value, Context context) throws IOException,
InterruptedException {
    String doc_id = key.toString();
    String text = value.toString();

    for (String pattern : skipList) {
        text = text.replaceAll(pattern, " ");
    }

    StringTokenizer itr = new StringTokenizer(text);
    String word;
    while (itr.hasMoreTokens()) {
        word = itr.nextToken();
        addResult(word, doc_id);
    }

    if ((++numRecords % 1000) == 0) {
        context.setStatus("Finished processing " + numRecords + " records");
        emitResults(context);
    }
  }

  private void loadSkipListFile(String skipListFile) {
   BufferedReader fis = null;
   try {
       fis = new BufferedReader(new FileReader(skipListFile));
       String pattern = null;
```

```
        while ((pattern = fis.readLine()) != null) {
          skipList.add(pattern);
        }
      } catch (IOException ioe) {
        System.err.println("Caught exception while loading skip file '" + skipListFile + "' : "
          + StringUtils.stringifyException(ioe));
      } finally {
        if (fis != null) {
          try {
            fis.close();
          } catch (IOException ioe) {
            System.err.println("Caught exception while closing skip file '" + skipListFile + "' : "
              + StringUtils.stringifyException(ioe));
          }
        }
      }
    }

    private void addResult(String word, String docid) {
      Map<String, Integer> counts = results.get(word);
      if (counts == null) {
        counts = new HashMap<String, Integer>();
        results.put(word, counts);
      }
      Integer count = counts.get(docid);
      if (count == null) {
        counts.put(docid, 1);
      } else {
        counts.put(docid, ++count);
      }
    }

    private void emitResults(Context context) throws IOException, InterruptedException {
      for (Entry<String, Map<String, Integer> > counts : results.entrySet()) {
        String word = counts.getKey();
        for (Entry<String, Integer> count : counts.getValue().entrySet()) {
          pair.set(word, count.getValue());
          docid.set(count.getKey());
          context.write(pair, docid);
        }
      }
      results.clear();
    }

    @Override
    public void cleanup(Context context) throws IOException, InterruptedException {
      emitResults(context);
    }
  }

  public static class IntSumReducer
      extends Reducer<MPair, Text, Text, Text> {
```

```
  //private IntWritable result = new IntWritable();
  private Map<String, Map<String, Integer> > results = new HashMap<String, Map<String,
Integer> >();
  private Map<String, Long> DD = new HashMap<String, Long>();
  //private final MPair pair = new MPair();
  private final Text word = new Text();
  private final Text docid = new Text();
  private long numRecords = 0;
  static double D;

  @Override
  public void setup(Context context) throws IOException, InterruptedException {
    D = Double.parseDouble(context.getConfiguration().get("buildinvertedindex.D"));
  }

  public void reduce(MPair key, Iterable<Text> values,
                     Context context
                     ) throws IOException, InterruptedException {
    String docid = "";
    for (Text val : values) {
      docid = val.toString();
    }

    addResult(key.getWord(), docid, key.getTf());
    //context.write(new Text(key.getWord() + " " + key.getTf()), new Text(docid));
    if ((numRecords % 1000) == 0) {
        context.setStatus("Reduce: Finished processing " + numRecords + " records");
        emitResults(context);
    }
  }

  private void addResult(String word, String docid, Integer tf) {
    Map<String, Integer> counts = results.get(word);
    if (counts == null) {
        counts = new LinkedHashMap<String, Integer>();
        results.put(word, counts);
        ++numRecords;
    }

    Integer count = counts.get(docid);
    if (count == null) {
        if (counts.size() < 20) {
          counts.put(docid, tf);
        }
    } else {
        counts.put(docid, count + tf);
    }

    Long dd = DD.get(word);
    if (dd == null) {
        DD.put(word, (long)tf);
    } else {
```

```
        DD.put(word, dd + tf);
      }
    }

    private void emitResults(Context context) throws IOException, InterruptedException {
      for (Entry<String, Map<String, Integer> > counts : results.entrySet()) {
        String wr = counts.getKey();
        word.set(wr);

        Long dd = DD.get(wr);
        Double idf = Math.abs(Math.log(D/dd));
        String text = /*dd.toString() + " : " + idf.toString() +*/ " [";
        for (Entry<String, Integer> count : counts.getValue().entrySet()) {
          text += "<" + count.getKey() + ", " + Double.valueOf(count.getValue() * idf).toString() +
">, ";
        }
        text = text.substring(0, text.length() - 2);
        text += "]";
        docid.set(text);
        context.write(word, docid);
      }
      results.clear();
      DD.clear();
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
      emitResults(context);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 4) {
      System.err.println("Usage: buildinvertedindex <in> <out> <skip_list> <d>");
      System.exit(2);
    }

    File skipFile = new File(otherArgs[2]);
    conf.set("buildinvertedindex.skip-list", skipFile.getName());
    conf.set("tmpfiles", "file://" + skipFile.getAbsolutePath());
    conf.set("buildinvertedindex.D", otherArgs[3]);

    conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", " ");
    conf.setBoolean("exact.match.only", true);
    conf.set("io.serializations",
          "org.apache.hadoop.io.serializer.JavaSerialization,"
          + "org.apache.hadoop.io.serializer.WritableSerialization");

    Job job = new Job(conf, "build inverted index");
    job.setInputFormatClass(XmlInputFormat.class);
```

```
    job.setJarByClass(BuildInvertedIndex.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setMapOutputKeyClass(MPair.class);
    job.setMapOutputValueClass(Text.class);

    //job.setCombinerClass(IntSumReducer.class);

    job.setPartitionerClass(MPartitioner.class);

    job.setSortComparatorClass(MGroupComparator.class);
    //job.setGroupingComparatorClass(MGroupComparator.class);

    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```