Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Отчёт по практической работе № 4

По дисциплине: «Распределенная обработка информации»

Тема: «MapReduce: инвертированный индекс»

Выполнил:
студент гр. МГ-101
Лукошкин В. Ю.

Проверил:
Профессор Кафедры ВС
Курносов М.Г.

Новосибирск 2021

Задание.

Имеется дамп Википедии в формате XML (enwiki.xml). **Требуется:**

1. Определить TOP-20 высокочастотных слов.
2. Посчитать количество документов.
3. Построить инвертированный индекс.

Ход работы.

Для начала необходимо посчитать количество каждого слова, встречающегося в дампе.

Копируем enwiki.xml в HDFS:

```
[lukoshkin@oak build-inverted-index]$ hdfs dfs -put enwiki.xml  /inverted-index/enwiki.xml
put: `/inverted-index/enwiki.xml': File exists
[lukoshkin@oak build-inverted-index]$
```

Компилируем WordCount.java, запускаем ./start-job.sh. Получившиеся результаты:

```
ы       2
ыран&quot;      1
экзотика        1
экипировке.JPG|thumb|right|Paratroopers 1
экономическом   1
экосистем       1
экрана  1
экс-губернатор  1
экспериментальной       1
электродистанционной    1
электросервопривода     1
эмиграция|date=20       1
энергоблокаа    1
энциклопедия)].&lt;/ref&gt;     1
энциклопедия}}),        1
энцыкляпэдыя    1
эпатаж&quot;?|work=BBC  1
эпоху|url=https://books.google.com/books?id=edsGAAAAYAAJ|year=1908|publisher=Тип.       1
эртний  1
эскортом        1
эстонцам.jpg|thumb|[[Vorkuta    1
этапы   1
этнического     1
этнической      2
этнографию      1
этноконфессиональный    1
этноса, 1
этого,  2
ю´      1
южных   1
юпсилон 1
ют      1
я       12
я&quot; 1
я...    1
ядерной 1
язык|Русский    1
языка]  2
языке|url=http://demoscope.ru/weekly/2013/0571/tema03.php|url-status=live|archive-url=https://web.archive.org/web/20140805082906/http://demoscope.ru/weekly/2013/0571/tema03.php|archive-date=5
языки]  2
языкознания     2
языкознания.    2
языку|url=http://demoscope.ru/weekly/2013/0571/tema02.php|work=[[Демоскоп       1
языкъ   1
якдигар 1
яку     1
января  14
января. 1
ястангуудын     1
ідиш,   1
ін.     1
інфраструктури  1
інш.;   1
інш].   4
історичному     1
історія 2
історії 3
іхньої  1
j),     1
давним  1
дазик,  1
дазик|trans-title=History       1
дазик}},        1
```

1. Определить TOP-20 высокочастотных слов.

Компилируем WordCountTop.java, запускаем ./start-job.sh. Получившиеся результаты:

```
\P{L}+
\sare\s
\sfrom\s
\son\s
\sthat\s
\swith\s
\sby\s
\swas\s
\sfor\s
\sas\s
\sThe\s
\sis\s
\s*\s
\s=\s
\sa\s
\sto\s
\s|\s
\sin\s
\sand\s
\sof\s
\sthe\s
[lukoshkin@oak wordcount-top-xml]$ hdfs dfs -cat ./inverted-index/wordcount-top-xml/output/part*
```

2. Посчитать количество документов.

Компилируем CountDocs.java, запускаем ./start-job.sh. Получившиеся результаты:

```
27432
[lukoshkin@oak count-docs-xml]$ hdfs dfs -cat ./inverted-index/count-docs-xml/output/part*
```

3. Построить инвертированный индекс.

Результирующий инвертированный индекс должен иметь следующую структуру: (word, [<docid1, TF-IDF1>, <docid2, TF-IDF2>, ...]). Статьи должны быть отсортированы в порядке убывания TF-IDF. Также отфильтрованы слова из списка TOP-20.

Компилируем BuildInvertedIndex.java, запускаем ./start-job.sh. Получившиеся

результаты:


lukoshkin@oak:~/lab4/inverted_index/build-inverted-index
Mamertino [<20314, 2.70805020110221>]
ManhattanTheCity [<18315, 2.70805020110221>]
Manazir [<1645, 2.008291961827888>, <39990, 0.22314355131420976>, <23289, 0.22314355131420976>, <26833, 0.22314355131420976>]
Maleficarum [<20561, 104.07606332328638>, <7482, 5.203803166164319>, <26154, 5.203803166164319>, <33959, 5.203803166164319>, <27165, 5.203803166164319>, <15191, 3.4692021107762128>, <24643, 3.4692021107762128>, <27796, 3.4692021107762128>, <36137, 1.7346010553881064>, <7324, 1.7346010553881064>, <27706, 1.7346010553881064>, <18836, 1.7346010553881064>, <27694, 1.7346010553881064>, <14068, 1.7346010553881064>, <13451, 1.7346010553881064>]
Manavar [<18606, 2.70805020110221>]
Manassites [<28179, 2.70805020110221>]
Mame [<19029, 2.772588722239781>, <6787, 2.0794415416798357>, <20572, 2.0794415416798357>, <10409, 2.0794415416798357>, <316, 1.3862943611198906>, <20889, 1.3862943611198906>, <16045, 1.3862943611198906>, <3846, 1.3862943611198906>, <11866, 0.6931471805599453>, <19296, 0.6931471805599453>, <5003, 0.6931471805599453>, <8957, 0.6931471805599453>, <21342, 0.6931471805599453>, <11587, 0.6931471805599453>, <34760, 0.6931471805599453>, <28271, 0.6931471805599453>, <8219, 0.6931471805599453>]
Mandaeaism [<19522, 2.70805020110221>]
Maneabau [<16678, 2.70805020110221>]
Malipop [<19130, 2.70805020110221>]
MaloryTowers [<10258, 4.828313737302301>]
ManichaeanElectaeKocho [<19760, 2.70805020110221>]
Mama [<16245, 35.47175539540405>, <15319, 27.870664953531758>, <18761, 25.336968139574324>, <4544, 22.803271325616894>, <20038, 12.668484069787162>, <31341, 10.13478725582973>, <25286, 7.601090441872298>, <1566, 7.601090441872298>, <33683, 7.601090441872298>, <28480, 7.601090441872298>, <3744, 7.601090441872298>, <28820, 7.601090441872298>, <16802, 7.601090441872298>, <6347, 7.601090441872298>, <39668, 5.067393627914865>, <4833, 5.067393627914865>, <11181, 5.067393627914865>, <37731, 5.067393627914865>, <25866, 5.067393627914865>, <7890, 5.067393627914865>]
Malukas [<27934, 4.828313737302301>]
Mammeri [<4693, 2.0149030205422647>, <358, 2.0149030205422647>]
Malakás [<39029, 2.70805020110221>]
Malavi [<19088, 4.828313737302301>]
Maldiviana [<19117, 2.70805020110221>]
Mami [<18175, 1.1507282898071236>, <2085, 0.5753641449035618>, <19955, 0.5753641449035618>, <12734, 0.2876820724517809>, <3092, 0.2876820724517809>, <20312, 0.2876820724517809>, <16433, 0.2876820724517809>, <8860, 0.2876820724517809>, <16873, 0.2876820724517809>, <10221, 0.2876820724517809>, <16175, 0.2876820724517809>, <28148, 0.2876820724517809>, <32669, 0.2876820724517809>, <23477, 0.2876820724517809>, <29789, 0.2876820724517809>]
Manapa [<37548, 1.3217558399823195>, <30059, 1.3217558399823195>, <24132, 1.3217558399823195>, <594, 1.3217558399823195>]
Malfertheiner [<22742, 2.0149030205422647>, <32323, 2.0149030205422647>]
Malato [<21444, 2.0149030205422647>, <18247, 2.0149030205422647>]
Malloch [<16844, 7.259370033829361>, <37107, 2.177811010148808>, <7044, 2.177811010148808>, <27650, 1.4518740067658722>, <6852, 1.4518740067658722>, <38172, 1.4518740067658722>, <4402, 0.7259370033829361>, <24408, 0.7259370033829361>, <32851, 0.7259370033829361>, <6140, 0.7259370033829361>, <25874, 0.7259370033829361>, <30273, 0.7259370033829361>, <20826, 0.7259370033829361>, <9032, 0.7259370033829361>, <14539, 0.7259370033829361>]
Manipuris [<9992, 1.6094379124341003>, <3454, 1.6094379124341003>, <10312, 1.6094379124341003>]
Malausène [<24060, 2.70805020110221>]
Manang [<21324, 2.1972245773362196>, <17168, 2.1972245773362196>, <37208, 1.0986122886681098>]
Maneouvering [<15068, 2.70805020110221>]
Malocco [<6611, 2.70805020110221>]
Mandolesi [<31880, 2.0149030205422647>, <12558, 2.0149030205422647>]
Manifestaartnet [<5593, 2.70805020110221>]
Mamaroneck [<9824, 4.029806041084529>]
Maldras [<23033, 4.029806041084529>]
Manasollasa [<37522, 2.70805020110221>]
Makkena [<34953, 2.70805020110221>]
Makovec [<21488, 2.70805020110221>]
Malmöit [<19021, 2.70805020110221>]
Manchuria [<25310, 66.48509080117002>, <32927, 39.28664456432774>, <16772, 33.24254540058501>, <14097, 27.19844623684228>, <16749, 27.19844623684228>, <15443, 24.176396654970915>, <34488, 21.15434707309955>, <17926, 18.132297491228186>, <29269, 18.132297491228186>, <14115, 15.110247909356822>, <21256, 15.110247909356822>, <6859, 15.110247909356822>, <27022, 15.110247909356822>, <17865, 15.110247909356822>, <31769, 12.088198327485458>, <16756, 12.088198327485458>, <37699, 12.088198327485458>, <5760, 12.088198327485458>, <27019, 12.088198327485458>, <21255, 9.066148745614094

Исходный код программы

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);
  }
}
```