Hadoop MapReduce
Инфраструктура за електронско пословање

др Милош ЦВЕТАНОВИЋ
др Захарије РАДИВОЈЕВИЋ

Hadoop

Софтверска библиотека Apache Hadoop представља оквир који омогућава дистрибуирану обраду великих скупова података коришћењем кластера рачунара и једноставног програмског модела. Омогућава скалирање апликација у зависности од доступног хардвера на коме је подигнут (и на системима са једним сервером и са великим бројем сервера код којих сваки има локални простор). Библиотека је дизајниран тако да на апликативном нивоу открије и обради кварове, што захтева коришћење наменског хардвера да би се постигла велика доступност, већ се може користити доступан хардвер.

Садржи следеће модуле:

- **Hadoop Common**: Модул садржи заједничке делове које користе остали модули.
- **Hadoop Distributed File System (HDFS™)**: Дистрибуирани фајл систем који омогућава високо-пропусни приступ подацима апликације.
- **Hadoop YARN**: Оквир за распоређивање послова и управљање ресурсима кластера.
- **Hadoop MapReduce**: Систем за паралелно процесирање великих скупова података (заснован на YARN модулу)

http://hadoop.apache.org/

MapReduce

•Мапридјус (MapReduce) представља програмски модел који треба да олакша обраду и креирање велике количине података користећи паралелне алгоритме који могу да се извршавају на групи рачунара. Инспирација потиче од функционалних језика (Lisp) који имају фазе мапирања и редукције.

•Коришћење се може поделите на следеће кораке:

1. Комплетан улазни сет података који је потребно да се обради се разбија на парове кључ-вредност.
2. У фазу мапирања као улазни податак пристиже један пар кључ-вредност који се обрађује. На основу овог пара података формира се листа која садржи нула или више парова који садрже кључ-вредност.
   Map: (key1, val1) -> listOf(key2, val2)
3. Када се заврши фаза мапирања излази фазе мапирања се групишу по вредности кључа и прослеђују у фазу редукције.
4. У фазу редукције као улазни податак пристиже један кључ и листа вредности које су за дати кључ креиране у фази мапирања. На основу ових вредности се формира листа резултата.
   Reduce: (key2, listOf(val2)) -> listOf(val3)

Ко све користи MapReduce и за шта?

- Google: (полазна основа, не користи Hadoop)
  - Конструкција индекса код претраживања страница (Google Search)
  - Груписање чланака (Google News)
  - Аутоматско превођење засновано на статистици
  - Код вишеслојних мапа улица
- Yahoo!:
  - Код претражива и индексирања страница (“Web map” Yahoo! Search)
  - Филтрирање спам порука (Yahoo! Mail)
- Facebook:
  - Обрада података (Data mining)
  - Оптимизација реклама
  - Филтрирање спама

Класа!

Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

Modifier and Type	Method and Description
protected void	cleanup(org.apache.hadoop.mapreduce.Mapper.Context context) Called once at the end of the task.
protected void	map(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Mapper.Context context) Called once for each key/value pair in the input split.
void	run(org.apache.hadoop.mapreduce.Mapper.Context context) Expert users can override this method for more complete control over the execution of the Mapper.
protected void	setup(org.apache.hadoop.mapreduce.Mapper.Context context) Called once at the beginning of the task.

map

```
protected void map(KEYIN key,
        VALUEIN value,
        org.apache.hadoop.mapreduce.Mapper.Context context)
             throws IOException,
                    InterruptedException
```

Called once for each key/value pair in the input split. Most applications should override this, but the default is the identity function.

Throws:

IOException

InterruptedException

Класа!

Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

Modifier and Type	Method and Description
protected void	cleanup(org.apache.hadoop.mapreduce.Reducer.Context context) Called once at the end of the task.
protected void	reduce(KEYIN key, Iterable<VALUEIN> values, org.apache.hadoop.mapreduce.Reducer.Context context) This method is called once for each key.
void	run(org.apache.hadoop.mapreduce.Reducer.Context context) Advanced application writers can use the run(org.apache.hadoop.mapreduce.Reducer.Context) method to control how the reduce task works.
protected void	setup(org.apache.hadoop.mapreduce.Reducer.Context context) Called once at the start of the task.

reduce

```
protected void reduce(KEYIN key,
          Iterable<VALUEIN> values,
          org.apache.hadoop.mapreduce.Reducer.Context context)
               throws IOException,
                      InterruptedException
```

This method is called once for each key. Most applications will define their reduce class by overriding this method. The default implementation is an identity function.

Throws:

IOException
InterruptedException

Reducer

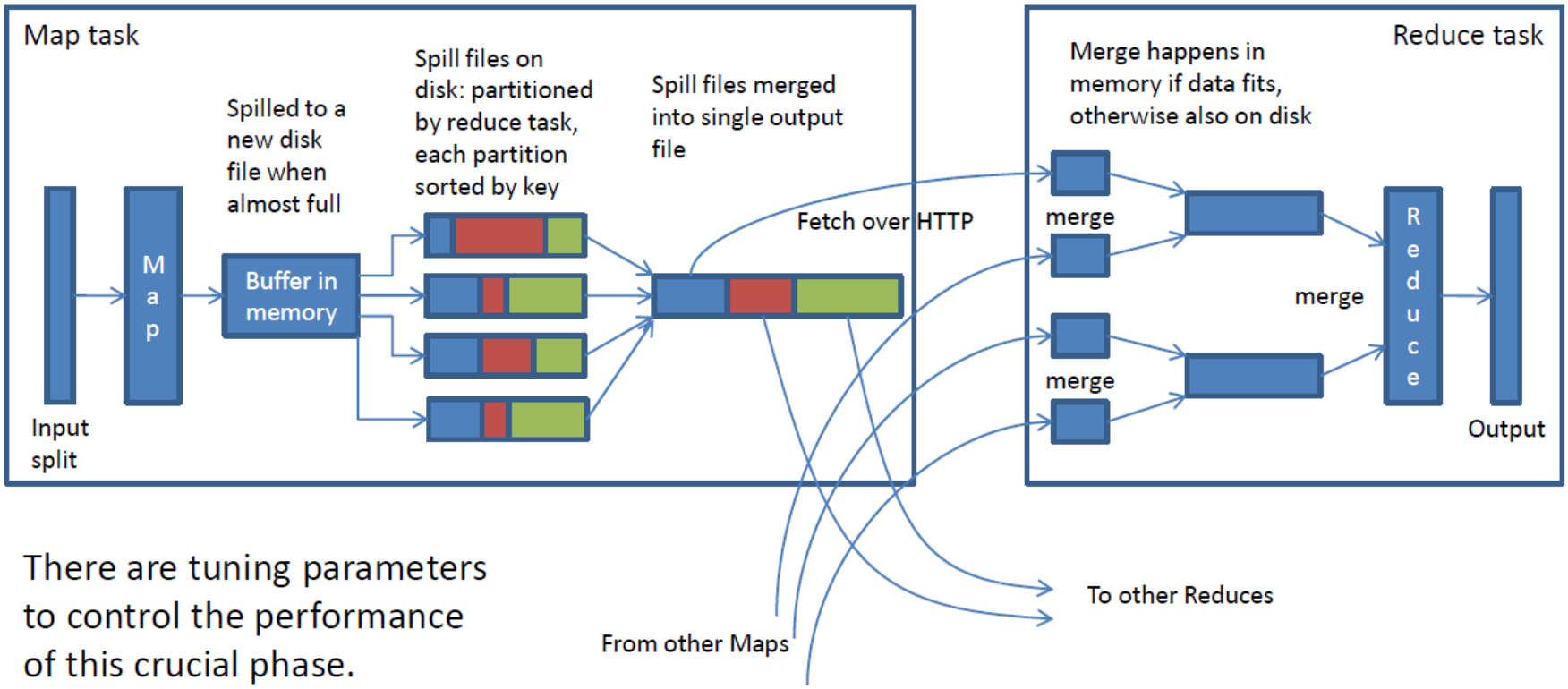
Редукција се састоји из три основне фазе:

1. **Мешање (Shuffle)**
   Приликом редукције се користи груписани излаз фазе мапирања. У овој фази систем, за сваки чвор који ради редукцију, проналази све релевантне делове излаза који су произвели чворови који раде мапирање. Проналажење се обавља користећи HTTP.
2. **Сортирање (Sort)**
   Систем групише улазе које треба редуковати користећи њихов кључ. Ово је потребно да се уради јер су различити чворово који раде мапирање могли за исти чвор који раде редукцију да произведу кључеве.
   Фаза мешања и сортирања се обављају симултано, то јест приликом дохватања излаза они се спајају. Додати за секундарно сортирање
3. **Редукција (Reduce)**
   У овој фази се обавља сама метода за редукцију reduce(Object, Iterable, Context) која се позива за сваки груписани пар <кључ, колекција вредности>.
   Излаз ове фазе се обично уписује у RecordWriter користећи TaskInputOutputContext.write(Object, Object).

Излаз фазе редукције се **не сортира.**

Мешање и сортирање

Поступак редукције започиње тако што се подаци ископирају из фазе мапирања чим буду доступни. Фаза редукције може да започне тек када се **комплетно** **заврши** фаза мапира и прикупе њени резултати.



There are tuning parameters to control the performance of this crucial phase.

Job

Modifier and Type	Method and Description
void	`setMapperClass(Class<? extends Mapper> cls)` Set the `Mapper` for the job.
void	`setReducerClass(Class<? extends Reducer> cls)` Set the `Reducer` for the job.
void	`setInputFormatClass(Class<? extends InputFormat> cls)` Set the `InputFormat` for the job.
void	`setOutputFormatClass(Class<? extends OutputFormat> cls)` Set the `OutputFormat` for the job.
void	`setOutputKeyClass(Class<?> theClass)` Set the key class for the job output data.
void	`setOutputValueClass(Class<?> theClass)` Set the value class for job outputs.
void	`setMapOutputKeyClass(Class<?> theClass)` Set the key class for the map output data.
void	`setMapOutputValueClass(Class<?> theClass)` Set the value class for the map output data.
void	`submit()` Submit the job to the cluster and return immediately.
boolean	`waitForCompletion(boolean verbose)` Submit the job to the cluster and wait for it to finish.

WordCount (1)

Написати MapReduce програм који чита садржај задате датотеке и пребројава колико се пута која реч у тој датотеци појавила.

```
public static class Map extends
        Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
```

WordCount (1)

Написати MapReduce програм који чита садржај задате датотеке и пребројава колико се пута која реч у тој датотеци појавила.

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

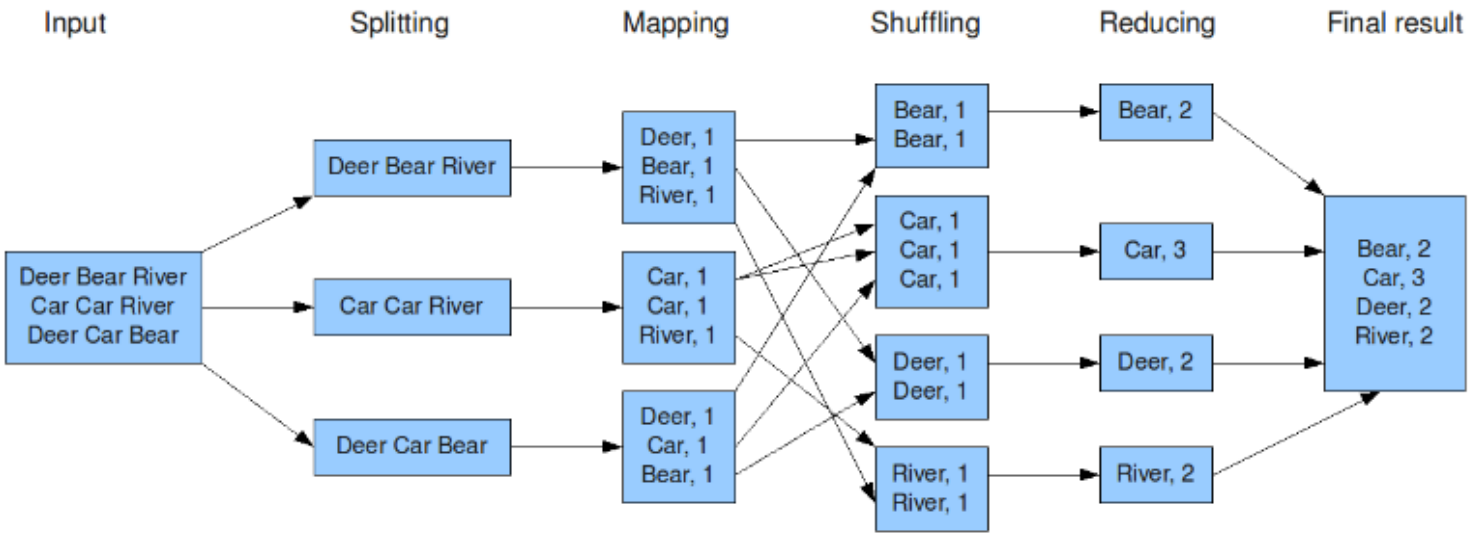
WordCount (1)

Написати MapReduce програм који чита садржај задате датотеке и пребројава колико се пута која реч у тој датотеци појавила.

```
public static void main(String[] args) throws Exception {

    Job job = Job.getInstance();
    job.setJarByClass(WordCount1.class);
    job.setJobName("wordcount1");

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}
```

WordCount (1)

Написати MapReduce програм који чита садржај задате датотеке и пребројава колико се пута која реч у тој датотеци појавила.



Configuration

Modifier and Type	Method and Description
void	setBoolean(String name, boolean value) Set the value of the name property to a boolean.
void	setDouble(String name, double value) Set the value of the name property to a double.
<T extends Enum<T>> void	setEnum(String name, T value) Set the value of the name property to the given type.
void	setFloat(String name, float value) Set the value of the name property to a float.
...	
String	get(String name, String defaultValue) Get the value of the name.
boolean	getBoolean(String name, boolean defaultValue) Get the value of the name property as a boolean.
double	getDouble(String name, double defaultValue) Get the value of the name property as a double.

WordCount (2)

У развијеном програму за пребројавање речи додати аргумент којим се задаје да ли води рачуна о малим и великим словима.

```
public static class Map extends
        Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    private boolean caseSensitive;
    public void setup(Context context) {
        caseSensitive = context.getConfiguration()
                .getBoolean("data", false);
    }
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        line = caseSensitive ? line : line.toLowerCase();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
```

WordCount (2)

У развијеном програму за пребројавање речи додати аргумент којим се задаје да ли води рачуна о малим и великим словима.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.setBoolean("data", true);

    Job job = Job.getInstance(conf, "wordcount2");

    job.setJarByClass(WordCount2.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}
```

Mapper.Context

Modifier and Type	Method and Description
`Counters.Counter`	`getCounter(Enum<?> name)` Get the `Counters.Counter` of the given group with the given name.
`Counters.Counter`	`getCounter(String group, String name)` Get the `Counters.Counter` of the given group with the given name.
`InputSplit`	`getInputSplit()` Get the `InputSplit` object for a map.
`float`	`getProgress()` Get the progress of the task.
`void`	`setStatus(String status)` Set the status description for the task.

WordCount (3)

У развијеном програму за пребројавање речи додати исписивање периодичног извештаја.

```
public static class Map extends
        Mapper<LongWritable, Text, Text, IntWritable> {
    ...
    private boolean caseSensitive;
    private String inputFile;
    private int numOfProcessedLines = 0;
    public void setup(Context context) {
        caseSensitive = context.getConfiguration().getBoolean("data", false);
        inputFile = ((FileSplit) context.getInputSplit()).getPath().getName();
    }
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        ...
        numOfProcessedLines++;
        int num = 0;
        while (tokenizer.hasMoreTokens()) {
            num++;
            word.set(tokenizer.nextToken());
            context.write(word, one);
            Counter c = context.getCounter(Counters.NUM_RECORDS);
            c.increment(1);
        }
        context.setStatus("" + numOfProcessedLines + ". in " + inputFile
                + " " + " were " + num + " words");
```

Комбиновање међурезултата

•Након обављања операције мапирања њени излази се налазе у оперативној меморији. Како би се повећала ефикасност обраде, да се **сви** подаци не би слсли на редукцију и губило време на паковање, транспорт и препакивање података, уведен је поступак комбиновања међурезултата. Овај поступак је сличан редукцији, с тим да се не мора да се обавља над свим подацима већ само над извесном количином података који су проистекли након мапирања.

•Приликом креирања треба водити рачуна да улазни/излазни кључ и вредност морају да буду истог топа као излази мапирања.

•Може применити само у случајевима када је посао таква да подржава инкременталну обраду

- MAX(5, 4, 1, 2) = MAX(MAX(5, 1), MAX(4, 2))
- Исто и за SUM, MIN, COUNT, ...

•Обично иста класа ради и мапирање и комбиновање резултата мапирања.

Job (наставак)

Modifier and Type	Method and Description
void	`setCombinerClass(Class<? extends Reducer> cls)` Set the combiner class for the job.
void	`setPartitionerClass(Class<? extends Partitioner> cls)` Set the `Partitioner` for the job.

WordCount (4)

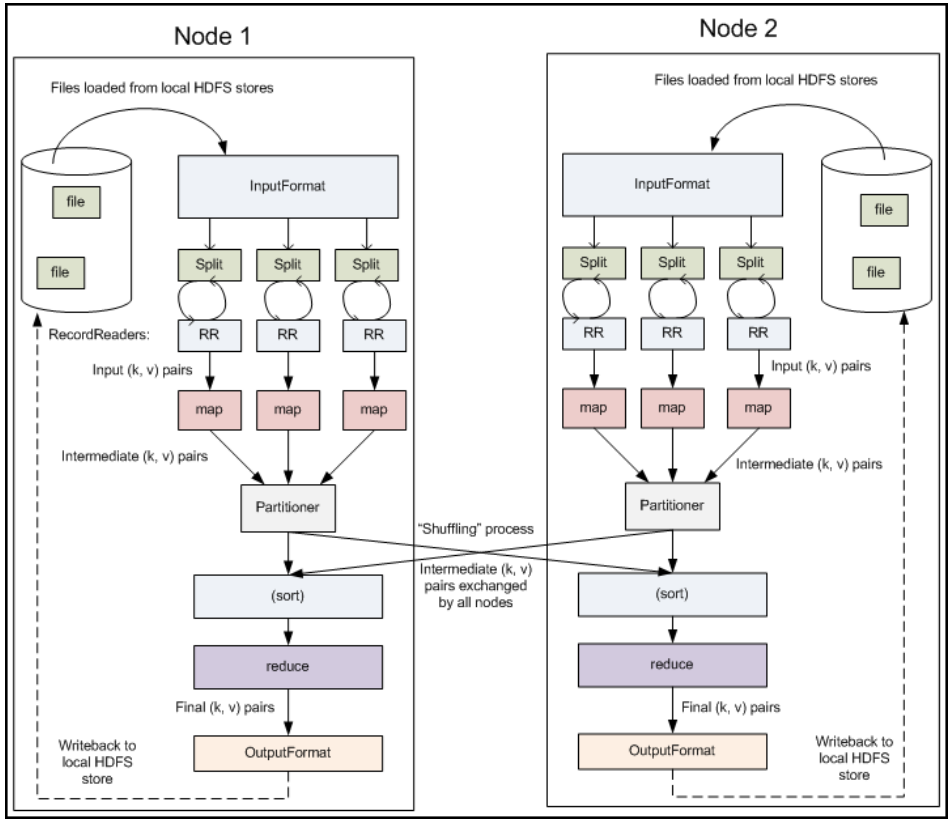
У развијеном програму за пребројавање речи додати комбиновање резултата на рачунару на коме се обавља мапирање.

```
public static class Reduce extends
        Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable value : values) {
            sum += value.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.setBoolean("data", true);

    Job job = Job.getInstance(conf, "wordcount2");

    job.setJarByClass(WordCount4.class);
    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);
```

Исти улазни и излазни типови!!

Дељење

•Класа Partitioner управља расподелом скупа кључева насталих након поступка мапирања различитим редукторима. Кључ, или његов подскуп, се користи да се скуп подели у групе. За ово поселу се обично користи хеш функција како би се кључеви равномерно расподелили. Број група за поделу је исти како и број редуктора. На овај начин се резултати мапирања деле међу редукторима.



Класа!

Partitioner<KEY,VALUE>

Modifier and Type	Method and Description
`abstract int`	`getPartition(KEY key, VALUE value, int numPartitions)` Get the partition number for a given key (hence record) given the total number of partitions i.e.

getPartition

```
public abstract int getPartition(KEY key,
                 VALUE value,
                 int numPartitions)
```

Get the partition number for a given key (hence record) given the total number of partitions i.e. number of reduce-tasks for the job.

Typically a hash function on a all or a subset of the key.

Parameters:

`key` - the key to be partioned.

`value` - the entry value.

`numPartitions` - the total number of partitions.

Returns:

the partition number for the `key`.

WordCount (5)

У развијеном програму за пребројавање речи додати распоређивање према хееш функцији.

```
public static class Partition extends Partitioner<Text, IntWritable> {

    @Override
    public int getPartition(Text key, IntWritable value, int numPartitions) {
        return key.hashCode() % numPartitions;
    }

}

public static void main(String[] args) throws Exception {
    ...
    job.setPartitionerClass(Partition.class);
    ...
}
```

Груписање резултата за редукцију

•Након дељења кључева редукторима могуће је њихово додатно сортирање по неком критеријуму користећи објекат за поређење који се поставља користећи методу setSortComparatorClass(Class) класе Job.

•Поред сортирања кључева могуће је њихово груписање како би се обавила обједињена обрада над груписаним кључевима. Ово се ради уколико се већи број кључева може груписати по неком свом својству у један кључ. За груписање кључева се користи посебан објекат за поређење RawComparator који се поставља користећи методу setGroupingComparatorClass (Class) класе Job. На овај начин се постиже ефекат који је доста сличан секундарном сортирању.

•Треба приметити да се прво обавља сортирање па груписање кључева.

Job (наставак)

Modifier and Type	Method and Description
void	`setGroupingComparatorClass(Class<? extends RawComparator> cls)` Define the comparator that controls which keys are grouped together for a single call to `Reducer.reduce(Object, Iterable, org.apache.hadoop.mapreduce.Reducer.Context)`
void	`setCombinerKeyGroupingComparatorClass(Class<? extends RawComparator> cls)` Define the comparator that controls which keys are grouped together for a single call to combiner, `Reducer.reduce(Object, Iterable, org.apache.hadoop.mapreduce.Reducer.Context)`
void	`setSortComparatorClass(Class<? extends RawComparator> cls)` Define the comparator that controls how the keys are sorted before they are passed to the `Reducer`.

Груписање резултата за редукцију - пример

Потребно је свакој страници (документу) придружити URL дупликата те странице који има највећи ранг.

- Кључ за Map фазу: url
- Вредност за Map: документ, ранг странице
- Излазни кључ Map фазе: чексума документа, ранг странице
- Излазна вредност Map фазе: url
- Дељење кључева између редуктора: по чексуми
- Сортирање кључева (SortComparator): по чексуми па по опадајућем рангу странице
- Груписање кључева (GroupingComparator): по чексуми

RawComparator<T>

compare

```
int compare(byte[] b1,
            int s1,
            int l1,
            byte[] b2,
            int s2,
            int l2)
```

Compare two objects in binary. b1[s1:l1] is the first object, and b2[s2:l2] is the second object.

Parameters:

`b1` - The first byte array.

`s1` - The position index in b1. The object under comparison's starting index.

`l1` - The length of the object in b1.

`b2` - The second byte array.

`s2` - The position index in b2. The object under comparison's starting index.

`l2` - The length of the object under comparison in b2.

Returns:

An integer result of the comparison.

Распоред обраде

1. Map.setup
2. Map.map(key1, value1)
3. if(hasMore keys1) goto 3
4. SortComparator.sort
5. Combiner.setup
6. Combiner.reduce
   при позиву values.next() позива се CombinerKeyGroupingComparator.compare
7. if(hasMore keys1) goto 6

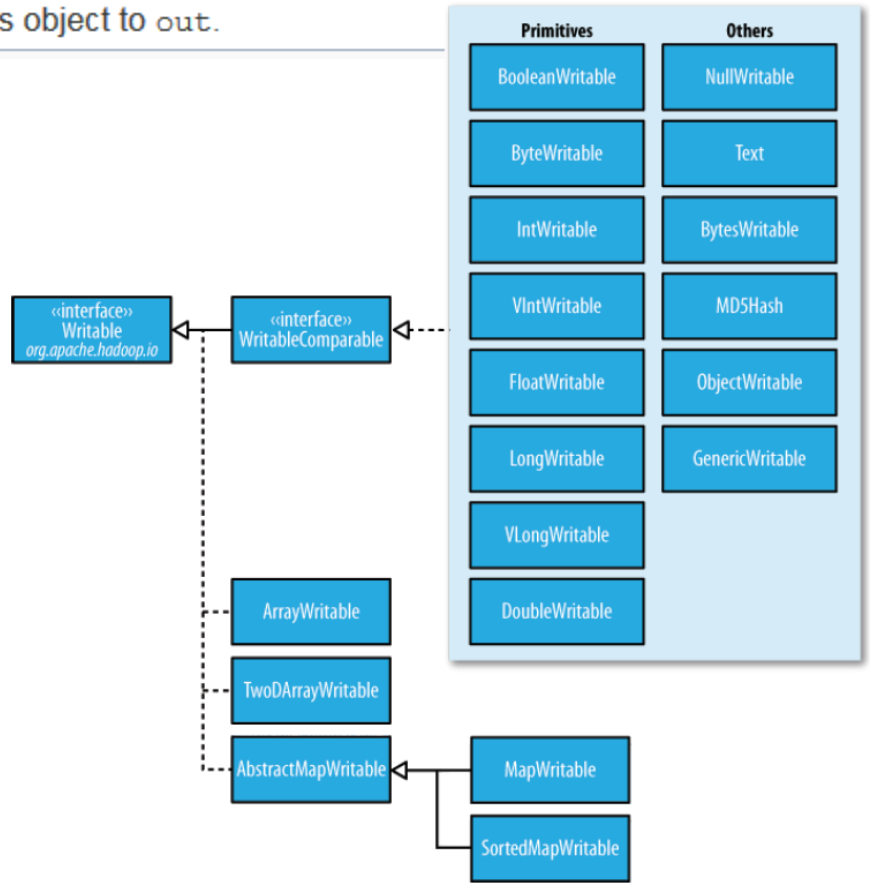
8. Partition.getPartition(key2, value2)

9. SortComparator.sort

10. Reduce.setup
11. Reducer.reduce(key, values),
    при позиву values.next() позива се GroupingComparator.compare
12. if(hasMore keys1) goto 11

Writable

Modifier and Type	Method and Description
`void`	`readFields(DataInput in)` Deserialize the fields of this object from `in`.
`void`	`write(DataOutput out)` Serialize the fields of this object to `out`.



WordCount (6)

У развијеном програму за пребројавање речи додати да пролази кроз више датотека, али да исписује за сваку датотеку колико се су пута појавиле речи које су исте дужине као најдужа реч у тој датотеци.

```
static class FileWord implements Writable {
    String fileName;
    String word;
    public void write(DataOutput out) throws IOException {
        Text.writeString(out, fileName);
        Text.writeString(out, word);
    }
    public void readFields(DataInput in) throws IOException {
        this.fileName = Text.readString(in);
        this.word = Text.readString(in);
    }
    public String toString() {
        String result = fileName + "\t" + word;
        return result;
    }
    public void extract(byte[] text, int start, int length) {
        try {

            byte[] ar1 = Arrays.copyOfRange(text, start, start + length);
            DataInputStream in1 = new DataInputStream(
                    new ByteArrayInputStream(ar1));
            readFields(in1);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

WordCount (6)

У развијеном програму за пребројавање речи додати да пролази кроз више датотека, али да исписује за сваку датотеку колико се су пута појавиле речи

```
public static class Map extends
        Mapper<LongWritable, Text, FileWord, IntWritable> {

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

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        line = caseSensitive ? line : line.toLowerCase();
        numOfProcessedLines++;
        StringTokenizer tokenizer = new StringTokenizer(line);
        int num = 0;
        while (tokenizer.hasMoreTokens()) {
            num++;
            FileWord out = new FileWord();
            out.fileName = inputFile;
            out.word = tokenizer.nextToken();
            context.write(out, one);
            Counter c = context.getCounter(Counters.NUM_RECORDS);
            c.increment(1);
        }
        context.setStatus("" + numOfProcessedLines + ". in " + inputFile
                + " " + " were " + num + " words");
```

WordCount (6)

У развијеном програму за пребројавање речи додати да пролази кроз више датотека, али да исписује за сваку датотеку колико се су пута појавиле речи које су исте дужине као најдужа реч у тој датотеци.

```
public static class Reduce extends
        Reducer<FileWord, IntWritable, FileWord, IntWritable> {

    String lastFileName = null;

    public void reduce(FileWord key, Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {

        if (lastFileName == null || !lastFileName.equals(key.fileName)) {
            lastFileName = key.fileName;
            int sum = 0;
            for (IntWritable value : values) {
                sum += value.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
}
```

WordCount (6)

У развијеном програму за пребројавање речи додати да пролази кроз више датотека, али да исписује за сваку датотеку колико се су пута појавиле речи које су исте дужине као најдужа реч у тој датотеци.

```
public static class Partition extends Partitioner<FileWord, IntWritable> {

    @Override
    public int getPartition(FileWord key, IntWritable value,
            int numPartitions) {
        return key.fileName.hashCode() % numPartitions;
    }

}
```

WordCount (6)

У развијеном програму за пребројавање речи додати да пролази кроз више датотека, али да исписује за сваку датотеку колико се су пута појавиле речи које су исте дужине као најдужа реч у тој датотеци.

```
public static final class GroupingComparator implements
        RawComparator<FileWord> {
    public int compare(byte[] text1, int start1, int length1, byte[] text2,
            int start2, int length2) {
        FileWord fw1 = new FileWord();
        fw1.extract(text1, start1, length1);
        FileWord fw2 = new FileWord();
        fw2.extract(text2, start2, length2);
        return compare(fw1, fw2);
    }

    public int compare(FileWord o1, FileWord o2) {
        int result = o1.fileName.compareTo(o2.fileName);
        if (result == 0) {
            result = -(o1.word.length() - o2.word.length());
        }
        return result;
    }
}
```

WordCount (6)

У развијеном програму за пребројавање речи додати да пролази кроз више датотека, али да исписује за сваку датотеку колико се су пута појавиле речи које су исте дужине као најдужа реч у тој датотеци.

```
public static final class SortComparator implements
        RawComparator<FileWord> {
    public int compare(byte[] text1, int start1, int length1, byte[] text2,
            int start2, int length2) {
        FileWord fw1 = new FileWord();
        fw1.extract(text1, start1, length1);
        FileWord fw2 = new FileWord();
        fw2.extract(text2, start2, length2);
        return compare(fw1, fw2);
    }

    public int compare(FileWord o1, FileWord o2) {
        int result = o1.fileName.compareTo(o2.fileName);
        if (result == 0) {
            result = -(o1.word.length() - o2.word.length());
        }
        if (result == 0) {
            result = o1.word.compareTo(o2.word);
        }
        return result;
    }
}
```

WordCount (6)

У развијеном програму за пребројавање речи додати да пролази кроз више датотека, али да исписује за сваку датотеку колико се су пута појавиле речи које су исте дужине као најдужа реч у тој датотеци.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.setBoolean("data", true);
    Job job = Job.getInstance(conf, "wordcount6");
    job.setJarByClass(WordCount5.class);
    job.setOutputKeyClass(FileWord.class);
    job.setOutputValueClass(IntWritable.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat
            .setInputPaths(job, new Path(args[0]), new Path(args[1]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setPartitionerClass(Partition.class);
    job.setGroupingComparatorClass(GroupingComparator.class);
    job.setSortComparatorClass(SortComparator.class);
    job.waitForCompletion(true);
}
```

Интеграл

Написати MapReduce програм који раруна интеграл дате функције у задатим границама.

```
public static class Map extends
        Mapper<LongWritable, Text, Text, DoubleWritable> {
    private Text word = new Text("Integral");
    private int numberOfIntervals;
    public void setup(Context context) {
        numberOfIntervals = context.getConfiguration().getInt("numberOfIntervals", 1000);
    }
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String[] args = line.split(",");
        double start = Double.parseDouble(args[0]);
        double end = Double.parseDouble(args[1]);
        double dx = (end - start) / numberOfIntervals;
        double x = start;
        double sum = 0;
        for (int i = 0; i < numberOfIntervals; i++) {
            sum = sum + f(x) * dx;
            x = x + dx;
        }
        DoubleWritable result = new DoubleWritable(sum);
        context.write(word, result);
    }
}
```

Интеграл

Написати MapReduce програм који раруна интеграл дате функције у задатим границама.

```
public static class Reduce extends
        Reducer<Text, DoubleWritable, Text, DoubleWritable> {

    public void reduce(Text key, Iterable<DoubleWritable> values,
            Context context) throws IOException, InterruptedException {
        double sum = 0;
        for (DoubleWritable val : values) {
            sum += val.get();
        }
        DoubleWritable result = new DoubleWritable(sum);
        context.write(key, result);
    }
}
```

Интеграл

Написати MapReduce програм који раруна интеграл дате функције у задатим границама.

```
private static double f(double x) {
    return x;
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.setInt("numberOfIntervals", 1000);
    Job job = Job.getInstance();
    job.setJarByClass(Integral.class);
    job.setJobName("Integral");
    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(DoubleWritable.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.waitForCompletion(true);
}
```

Множење матрица

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

```
map(key, value):
    // value is ("A", i, j, a_ij) or ("B", j, k, b_jk)
    if value[0] == "A":
        i = value[1]
        j = value[2]
        a_ij = value[3]
        for k = 1 to p:
            emit((i, k), (A, j, a_ij))
    else:
        j = value[1]
        k = value[2]
        b_jk = value[3]
        for i = 1 to m:
            emit((i, k), (B, j, b_jk))

reduce(key, values):
    // key is (i, k)
    // values is a list of ("A", j, a_ij) and ("B", j, b_jk)
    hash_A = {j: a_ij for (x, j, a_ij) in values if x == A}
    hash_B = {j: b_jk for (x, j, b_jk) in values if x == B}
    result = 0
    for j = 1 to n:
        result += hash_A[j] * hash_B[j]
    emit(key, result)
```

http://importantfish.com/one-step-matrix-multiplication-with-hadoop/

Множење матрица

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

```
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        Configuration conf = context.getConfiguration();
        int m = Integer.parseInt(conf.get("m"));
        int p = Integer.parseInt(conf.get("p"));
        String line = value.toString();
        String[] indicesAndValue = line.split(",");
        Text outputKey = new Text();
        Text outputValue = new Text();
        if (indicesAndValue[0].equals("A")) {
            for (int k = 0; k < p; k++) {
                outputKey.set(indicesAndValue[1] + "," + k);
                outputValue.set("A," + indicesAndValue[2] + ","
                        + indicesAndValue[3]);
                context.write(outputKey, outputValue);
            }
        } else {
            for (int i = 0; i < m; i++) {
                outputKey.set(i + "," + indicesAndValue[2]);
                outputValue.set("B," + indicesAndValue[1] + ","
                        + indicesAndValue[3]);
                context.write(outputKey, outputValue);
            }
        }
    }
}
```

http://importantfish.com/one-step-matrix-multiplication-with-hadoop/

Множење матрица

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

```
public static class Reduce extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {
            String[] value;
            HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();
            HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();
            for (Text val : values) {
                value = val.toString().split(",");
                if (value[0].equals("A")) {
                    hashA.put(Integer.parseInt(value[1]),
                            Float.parseFloat(value[2]));
                } else {
                    hashB.put(Integer.parseInt(value[1]),
                            Float.parseFloat(value[2]));
                }
            }
            int n = Integer.parseInt(context.getConfiguration().get("n"));
            float result = 0.0f;
            float a_ij;
            float b_jk;
            for (int j = 0; j < n; j++) {
                a_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
                b_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
                result += a_ij * b_jk;
            }
            if (result != 0.0f) {
                context.write(null,
                        new Text(key.toString() + "," + Float.toString(result)));
            }
```

http://importantfish.com/one-step-matrix-multiplication-with-hadoop/

Множење матрица

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    // A is an m-by-n matrix; B is an n-by-p matrix.
    conf.set("m", "2");
    conf.set("n", "5");
    conf.set("p", "3");

    Job job = new Job(conf, "MatrixMatrixMultiplicationOneStep");
    job.setJarByClass(OneStepMatrixMultiplication.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}
```

http://importantfish.com/one-step-matrix-multiplication-with-hadoop/

Множење матрица (2)

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

Корак 1:

```
map(key, value):
    // value is ("A", i, j, a_ij) or ("B", j, k, b_jk)
    if value[0] == "A":
        i = value[1]
        j = value[2]
        a_ij = value[3]
        emit(j, ("A", i, a_ij))
    else:
        j = value[1]
        k = value[2]
        b_jk = value[3]
        emit(j, ("B", k, b_jk))

reduce(key, values):
    // key is j
    // values is a list of ("A", i, a_ij) and ("B", k, b_jk)
    list_A = [(i, a_ij) for (M, i, a_ij) in values if M == "A"]
    list_B = [(k, b_jk) for (M, k, b_jk) in values if M == "B"]
    for (i, a_ij) in list_A:
        for (k, b_jk) in list_B:
            emit((i, k), a_ij*b_jk)
```

http://importantfish.com/two-step-matrix-multiplication-with-hadoop/

Множење матрица (2)

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

Корак 2:

```
map(key, value):
    emit(key, value)

reduce(key, values):
    result = 0
    for value in values:
        result += value
    emit(key, result)
```

http://importantfish.com/two-step-matrix-multiplication-with-hadoop/

Множење матрица (2)

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

```
public static class Map1 extends Mapper<LongWritable, Text, Text, Text> {
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String[] indicesAndValue = line.split(",");
        Text outputKey = new Text();
        Text outputValue = new Text();
        if (indicesAndValue[0].equals("A")) {
            outputKey.set(indicesAndValue[2]);
            outputValue.set("A," + indicesAndValue[1] + ","
                    + indicesAndValue[3]);
            context.write(outputKey, outputValue);
        } else {
            outputKey.set(indicesAndValue[1]);
            outputValue.set("B," + indicesAndValue[2] + ","
                    + indicesAndValue[3]);
            context.write(outputKey, outputValue);
        }
    }
}
```

http://importantfish.com/two-step-matrix-multiplication-with-hadoop/

Множење матрица (2)

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

```
public static class Reduce1 extends Reducer<Text, Text, Text, Text> {
    Text matrix = new Text("matrix");
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        String[] value;
        ArrayList<Entry<Integer, Float>> listA = new ArrayList<Entry<Integer, Float>>();
        ArrayList<Entry<Integer, Float>> listB = new ArrayList<Entry<Integer, Float>>();
        for (Text val : values) {
            value = val.toString().split(",");
            if (value[0].equals("A")) {
                listA.add(new SimpleEntry<Integer, Float>(Integer
                        .parseInt(value[1]), Float.parseFloat(value[2])));
            } else {
                listB.add(new SimpleEntry<Integer, Float>(Integer
                        .parseInt(value[1]), Float.parseFloat(value[2])));
            }
        }
        String i;
        float a_ij;
        String k;
        float b_jk;
        Text outputValue = new Text();
        for (Entry<Integer, Float> a : listA) {
            i = Integer.toString(a.getKey());
            a_ij = a.getValue();
            for (Entry<Integer, Float> b : listB) {
                k = Integer.toString(b.getKey());
                b_jk = b.getValue();
                outputValue.set(i + "," + k + ","
                        + Float.toString(a_ij * b_jk));
                context.write(matrix, outputValue);
            }
        }
```

http://importantfish.com/two-step-matrix-multiplication-with-hadoop/

Множење матрица (2)

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

```
public static void step1(String[] args) throws IOException,
        InterruptedException, ClassNotFoundException {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "MatrixMatrixMultiplicationTwoSteps");
    job.setJarByClass(TwoStepMatrixMultiplication.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    job.setMapperClass(Map1.class);
    job.setReducerClass(Reduce1.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path("temp"));

    job.waitForCompletion(true);
}
```

http://importantfish.com/two-step-matrix-multiplication-with-hadoop/

Множење матрица (2)

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

```
public static class Map2 extends Mapper<LongWritable, Text, Text, Text> {
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String[] indicesAndValue = value.toString().split("\t");
        String[] data = indicesAndValue[1].split(",");
        context.write(new Text(data[0] + "," + data[1]), new Text(data[2]));
    }
}

public static class Reduce2 extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        double sum = 0;
        Text result = new Text();
        for (Text text : values) {
            sum += Double.parseDouble(text.toString());
        }
        result.set(Double.toString(sum));
        context.write(key, result);
    }
}
```

http://importantfish.com/two-step-matrix-multiplication-with-hadoop/

Множење матрица (2)

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

```
public static void step2(String[] args) throws IOException,
        InterruptedException, ClassNotFoundException {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "MatrixMatrixMultiplicationTwoSteps2");
    job.setJarByClass(TwoStepMatrixMultiplication.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    job.setMapperClass(Map2.class);
    job.setReducerClass(Reduce2.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path("temp"));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}
```

http://importantfish.com/two-step-matrix-multiplication-with-hadoop/

Множење матрица (2)

Написати MapReduce програм који обавља множење матрица прочитаних из једне датотеке.

```
public static void main(String[] args) throws Exception {
    step1(args);
    step2(args);
}
```

http://importantfish.com/two-step-matrix-multiplication-with-hadoop/

Удаљеност чворова у графу

Написати MapReduce програм који у задатом графу тражи удаљеност између појединих чворова.

```
public static class Map1 extends Mapper<LongWritable, Text, Text, Pair> {
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String data[] = line.split("\t");
        Pair result1 = new Pair();
        result1.first.set(data[0]);
        result1.second.set(data[1]);
        if (data.length > 2) {
            result1.distance.set(Integer.parseInt(data[2]));
        } else {
            result1.distance.set(1);
        }
        context.write(result1.first, result1);
        Pair result2 = new Pair(result1.second, result1.first,
                result1.distance.get());
        context.write(result2.first, result2);

    }
}
```

Удаљеност чворова у графу

Написати MapReduce програм који у задатом графу тражи удаљеност између појединих чворова.

```
public static class Reduce1 extends Reducer<Text, Pair, Text, Pair> {
    public void reduce(Text key, Iterable<Pair> values, Context context)
            throws IOException, InterruptedException {
        Set<Pair> pairs = new HashSet<Pair>();
        for (Pair pair : values) {
            pairs.add(new Pair(pair.first, pair.second, pair.distance.get()));
            context.write(null, pair);
        }
        for (Pair a : pairs) {
            for (Pair b : pairs) {
                if (!a.second.equals(b.second)) {
                    Pair c1 = new Pair(a.second, b.second, a.distance.get()
                            + b.distance.get());
                    context.write(null, c1);
                    Pair c2 = new Pair(b.second, a.second, a.distance.get()
                            + b.distance.get());
                    context.write(null, c2);
                }
            }

        }
    }
}
```

Удаљеност чворова у графу

Написати MapReduce програм који у задатом графу тражи удаљеност између појединих чворова.

```
public static class Map2 extends Mapper<LongWritable, Text, Pair, Pair> {
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String data[] = line.split("\t");
        Pair result = new Pair();
        result.first.set(data[0]);
        result.second.set(data[1]);
        if (data.length > 2) {
            result.distance.set(Integer.parseInt(data[2]));
        } else {
            result.distance.set(1);
        }
        context.write(result, result);

    }
}
```

Удаљеност чворова у графу

Написати MapReduce програм који у задатом графу тражи удаљеност између појединих чворова.

```
public static class Reduce2 extends Reducer<Pair, Pair, Pair, Pair> {

    String lastPair = null;

    public void reduce(Pair key, Iterable<Pair> values, Context context)
            throws IOException, InterruptedException {
        String newPair = key.first.toString() + ":" + key.second.toString();
        if (lastPair == null || !lastPair.equals(newPair)) {
            lastPair = newPair;
            context.write(null, key);
        }
    }
}
```

Удаљеност чворова у графу

Написати MapReduce програм који у задатом графу тражи удаљеност између појединих чворова.

```
public static final class GroupingReducerComparator implements
        RawComparator<Pair> {
    public int compare(Pair o1, Pair o2) {
        int result = o1.first.compareTo(o2.first);
        if (result == 0) {
            result = o1.second.compareTo(o2.second);
        }
        return result;
    }
}

public static final class SortReducerComparator implements
        RawComparator<Pair> {
    public int compare(Pair o1, Pair o2) {
        int result = o1.first.compareTo(o2.first);
        if (result == 0) {
            result = o1.second.compareTo(o2.second);
        }
        if (result == 0) {
            result = o1.distance.get() - o2.distance.get();
        }
        return result;
    }
}
```

Удаљеност чворова у графу

Написати MapReduce програм који у задатом графу тражи удаљеност између појединих чворова.

```
public static void extractJob1(String src, String dst) throws IOException,
        InterruptedException, ClassNotFoundException {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "Distance1");
    job.setJarByClass(Distance.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Pair.class);

    job.setMapperClass(Map1.class);
    job.setReducerClass(Reduce1.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(src));
    FileOutputFormat.setOutputPath(job, new Path(dst));

    job.waitForCompletion(true);
}
```

Удаљеност чворова у графу

Написати MapReduce програм који у задатом графу тражи удаљеност између појединих чворова.

```
public static void extractJob2(String src, String dst) throws IOException,
        InterruptedException, ClassNotFoundException {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "Distance2");
    job.setJarByClass(Distance.class);
    job.setOutputKeyClass(Pair.class);
    job.setOutputValueClass(Pair.class);

    job.setMapperClass(Map2.class);
    job.setReducerClass(Reduce2.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(src));
    FileOutputFormat.setOutputPath(job, new Path(dst));

    job.setGroupingComparatorClass(GroupingReducerComparator.class);
    job.setSortComparatorClass(SortReducerComparator.class);
    job.waitForCompletion(true);
}
```

Удаљеност чворова у графу

Написати MapReduce програм који у задатом графу тражи удаљеност између појединих чворова.

```
public static void main(String[] args) throws Exception {
    String a;
    String b = args[0];
    for (int i = 0; i < 10; i++) {
        a = b;
        b = "temp\\job1\\" + i;
        extractJob1(a, b);
        a = b;
        b = "temp\\job2\\" + i;
        extractJob2(a, b);
    }
}
```

Корисне адресе:

1. The Apache Software Foundation, “Apache™ Hadoop®!,” http://hadoop.apache.org/
2. The Apache Software Foundation, “Туторијал за MapReduce,” http://hadoop.apache.org/docs/r2.7.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Додаци за коришћење Hadoop сервиса користећи Microsoft Azure сервис

1. Microsoft, “Емулатор за Hadoop,” http://www.microsoft.com/web/gallery/install.aspx?appid=HDINSIGHT
2. Microsoft, “HDInsight Service,” http://azure.microsoft.com/en-us/services/hdinsight/ (Hadoop за Microsoft Azure)
3. Microsoft, “Упутство за инсталацију додатка за Еклипс,” http://msdn.microsoft.com/en-us/library/azure/hh690946.aspx
4. Microsoft, “Windows Azure SDK,” http://go.microsoft.com/fwlink/?LinkID=252838