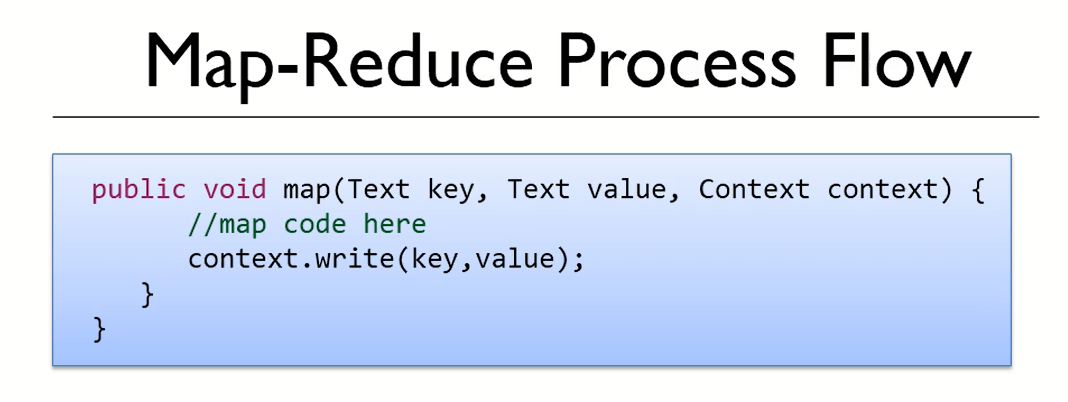
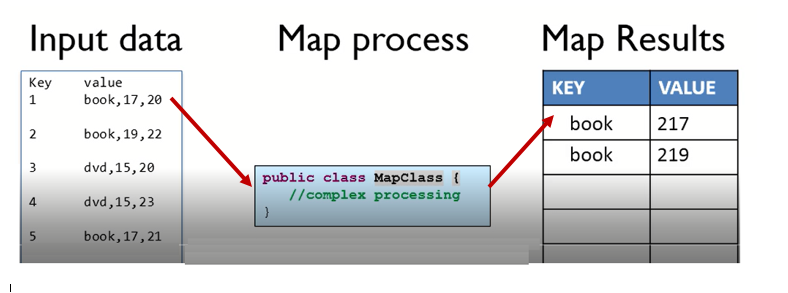
**Map-Reduse Shufle Step**

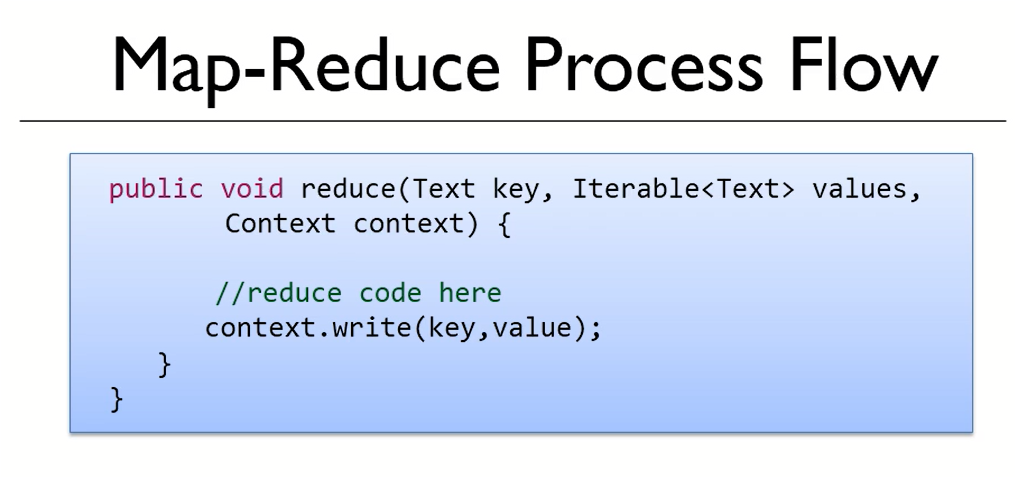


map მეთოდის შემავალი პარამეტრებია (Objekt key, Text value) და ამ მეთოდის შესრულების შედეგს, რომელიც წარმოდგენილია ასევე (key, value) წყვილით ვწერთ კონტექსტში Context.

ჩვენ შემთხვევაში როდესაც ვამუშავებდით ფაილის მონაცემებს map მეთოდის შემავალი პარამეტრებიდან key პარამეტრის მნიშვნელობა ავტომატურად განისაზღვრებოდა, როგორც ფაილში სტრიქონის ნომერი ხოლო value-ს როლში თავად სტრიქონი.



map მეთოდი ეშვება ყოველი შემავალი (key, value) წყვილისათვის და მის მუშაობის შედეგს წერს ასევე (key, value) ფორმატში

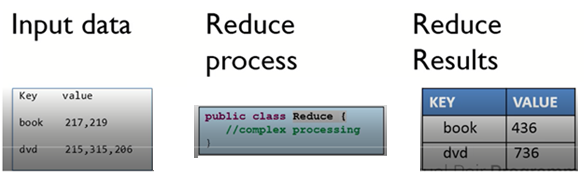


განსხვავებით map მეთოდისაგან reduce მეთოდის პარამეტრებს წარმოადგენენ

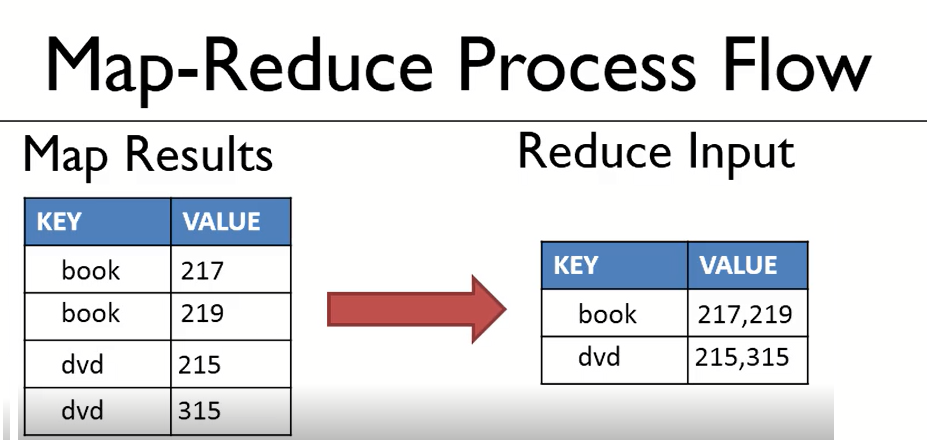


აღსანიშნავია რომ reduce მეტოდი ეშვება ერთხელ თითოეული Key-სათვის

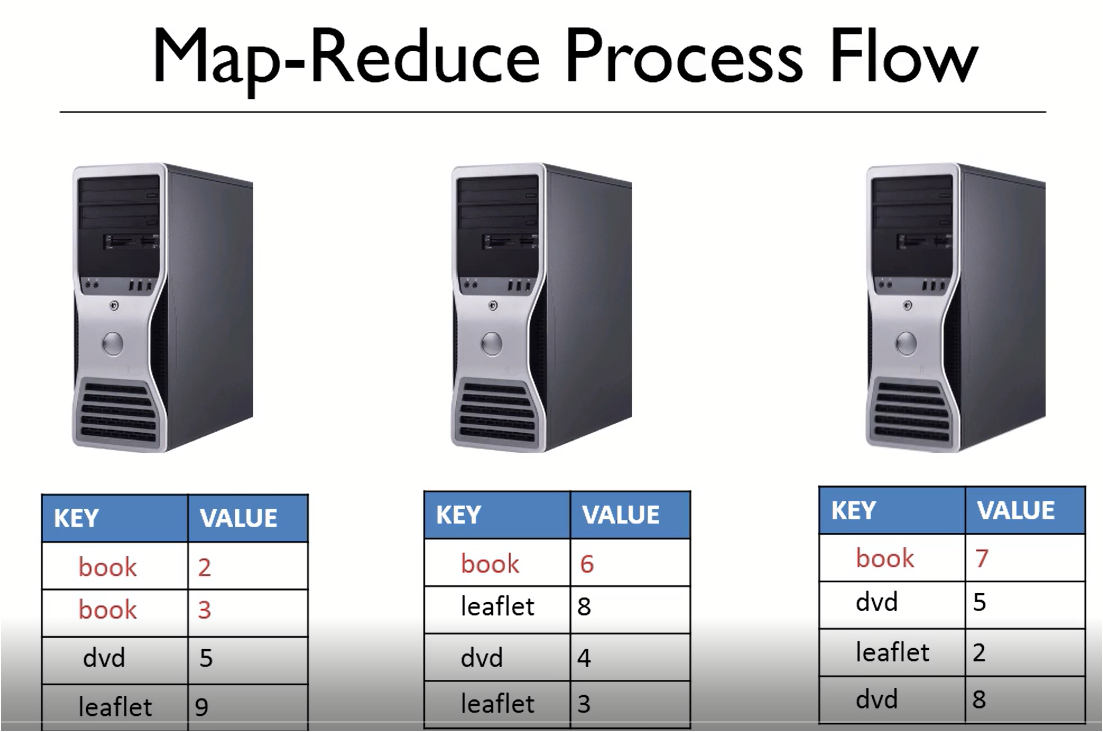
Reduce მეთოდის გაშვებამდე Hadoop აჯგუფებს ყველა განსხვაებულ Key-ს და იმ რაოდენობებს



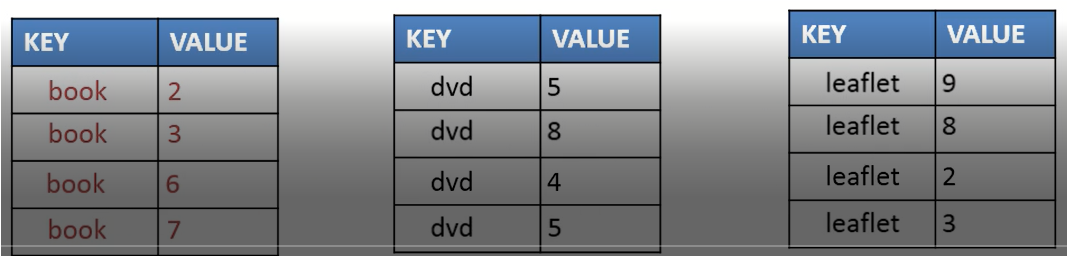
როგორც ვხედავთ map მეთოდის დასრულების შემდგომ ჩვენ გვაქვს ერთი დამატებითი საფეხური:



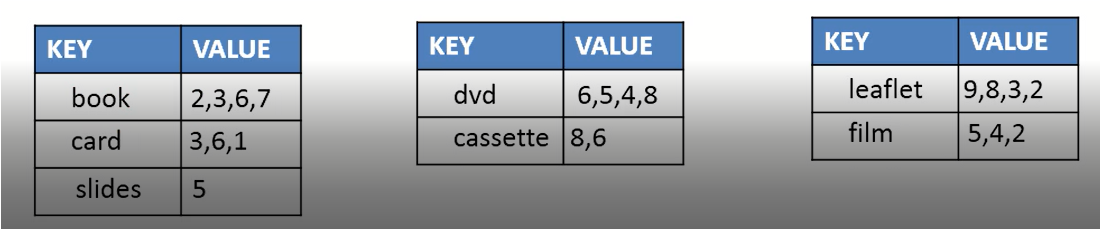
**როდესაც ჩვენ საქმე გვაქვს Hadoob-ის განაწილებულ კლასტერთან ასეთ შემთხვევაში თითოეულ Node-ზე map მეთოდის დასრულების შემდეგ გვექნება map Output**



მანამდე სანამ reduce მეთოდი გაეშვება საჭიროა რომ თითოეული გასაღების key-სათვის მნიშვნელობები დაჯგუფდეს თითოეულ Node-ზე:



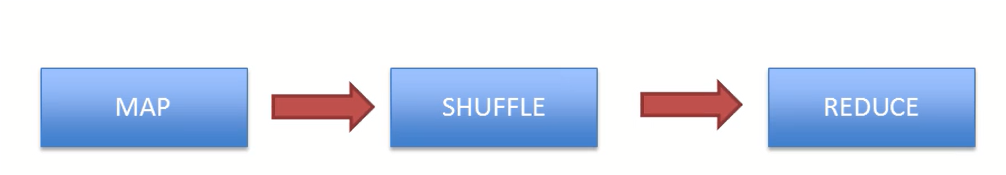
შემდგომ კი გარდაიქმნას შემდეგი სახით:



აღნიშნულ ოპერაციებს ახორციელებს Shufle პროცესი (საფეხური)

ამის შემდგომ კი შესაძლებელია reduce მეთოდის გაშვება თითოეულ ნოდზე

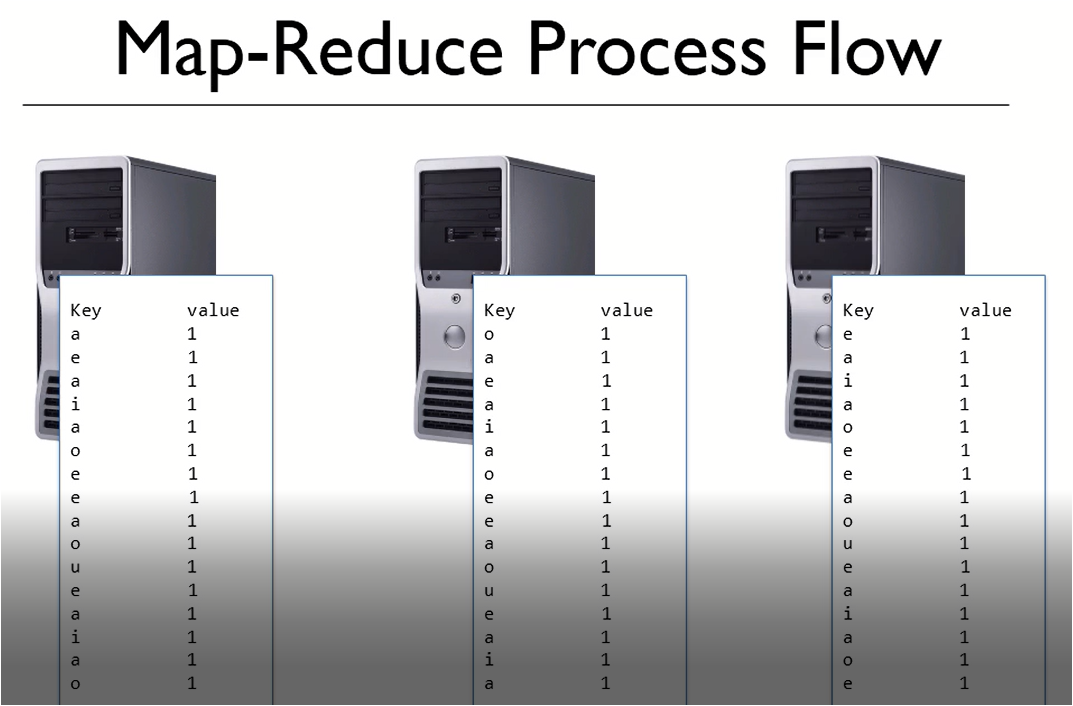
ამრიგად სინამდვილესი ჩვენ გვაქვს MapReduce პროცესის შემდეგი საფეხურები (ეტაპები):



SHUFFLE პროცესი არ იწყება მანამდე სანამ არ დასრულდება MAP პროცესი ხოლო

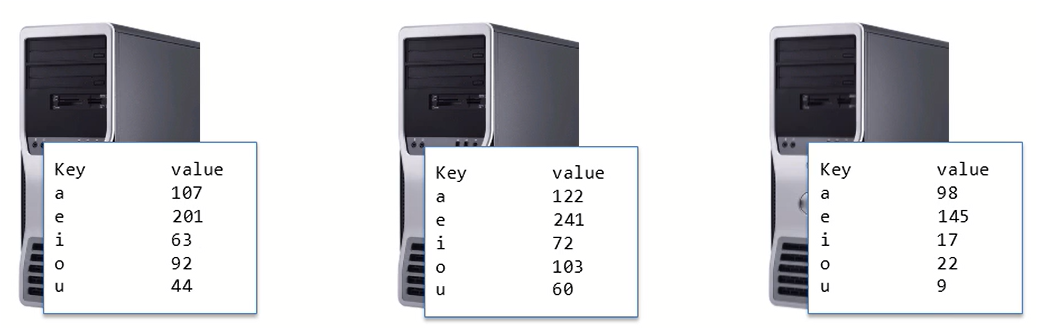
REDUCE პროცესი არ იწყება მანამდე სანამ არ დასრულდება SHUFFLE პროცესი.

დავუბრუნდეთ ხმოვნების რაოდენობების გამოთვლის ამოცანას. MAP პროცესის დასრულების შემდგომ ჩვენ გვექნება შემდეგი სურათი:

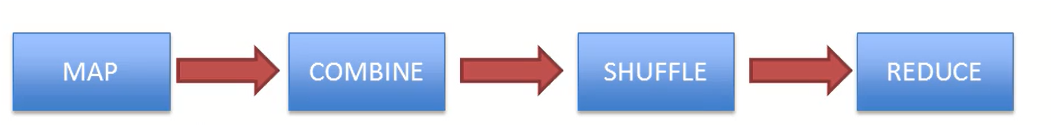


თუ ამ პროცესის შემდგომ ჩვენ დავიწყებდით მიღებული key, value წყვილების გადაჯგუფებას Hadoop-ის კლასტერის ხვადასხვა node-ებზე, ეს პროცესი გამოიწვევს ქსელში დიდი მონაცემების გადაცემას და შესაბამისად ქსელის დატვირთვას.

იმისათვის რომ თავიდან ავიცილოთ ქსელის გადატვირთვა საჭიროა SHUFFLE პროცესის დაწყებამდე მოვახდინოთ შემდეგი სახის დაჯგუფება:

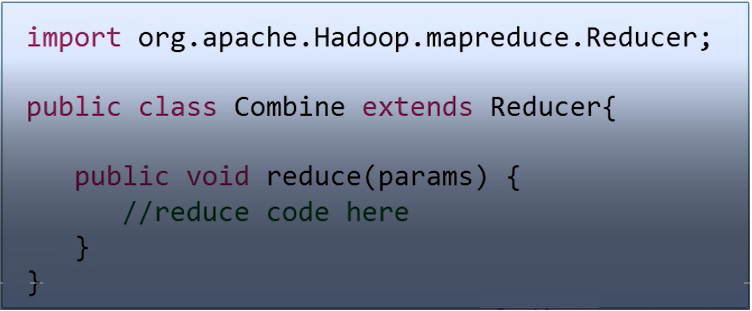


აღნიშნულს ეწოდება COMBINE- პროცესი. ამრიგად ჩვენ მივედით MapReduce პროცესის შემდეგ საფეხურებამდე:

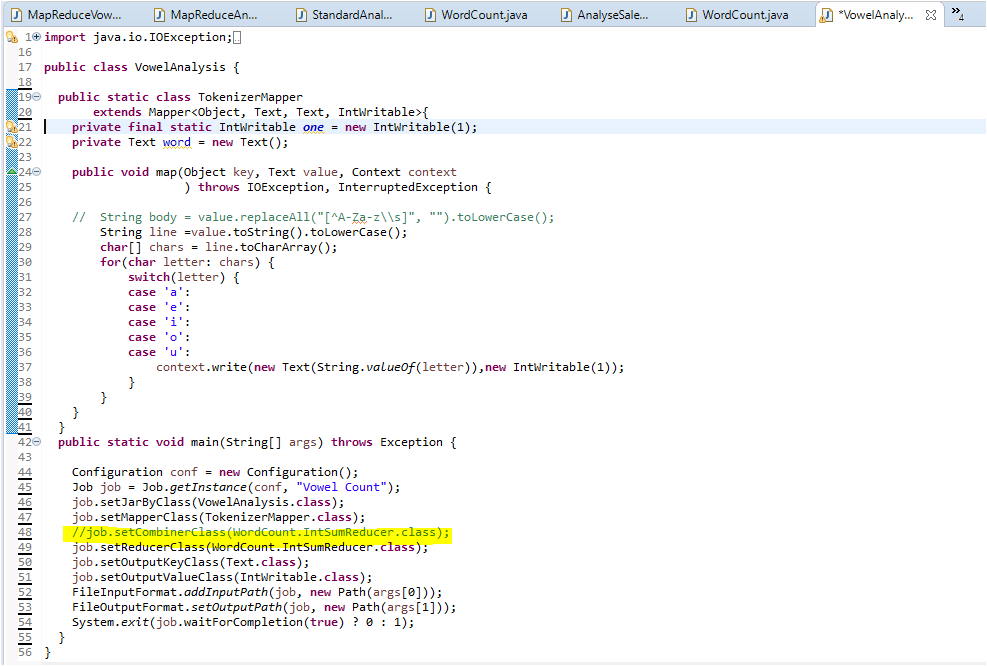


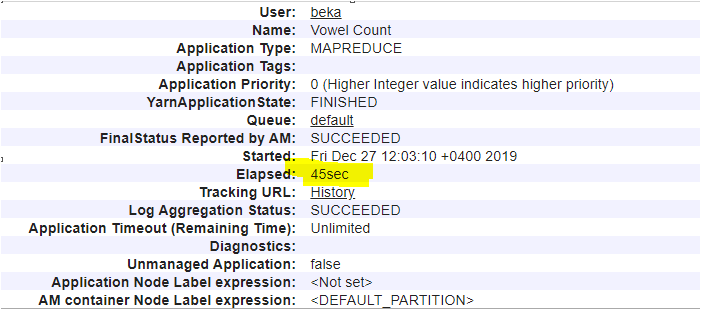
აღსანიშნავია ამასთან რომ COMBINE-საფეხურის სინტაქსი იგივეა რაც REDUCE საფეხურის სინტაქსი. მართლაც თითოეული მათთაგანი (როგორც REDUCE ასევე COMBINE) იღებს გასაღების (key-ის) მნიშვნელობას და აჯამებს შესაბამის value- მნიშვნელობებს.

ამრიგად ჩვენ შეგვიძლია გამოვიყენოთ ერთიდაიგივე პროგრამული კოდი როგორც REDUCE ასევე COMBINE საფეხურებისათვის

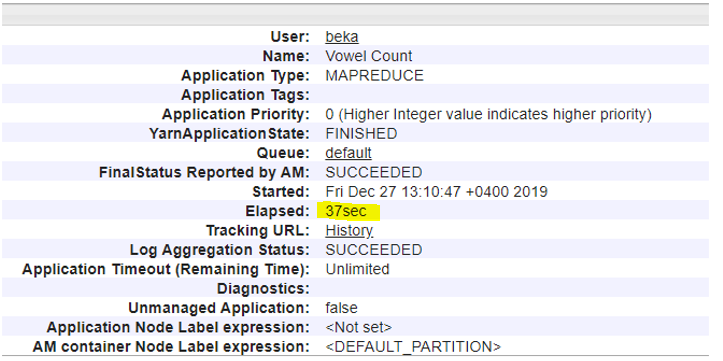


MapReduce without Combiner Step

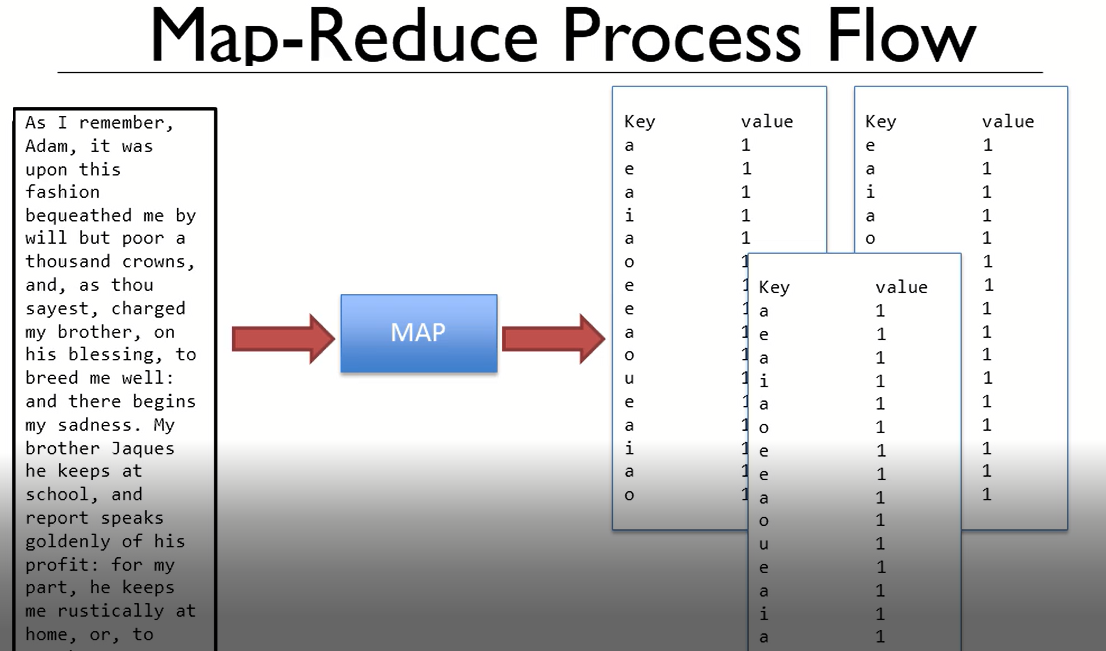


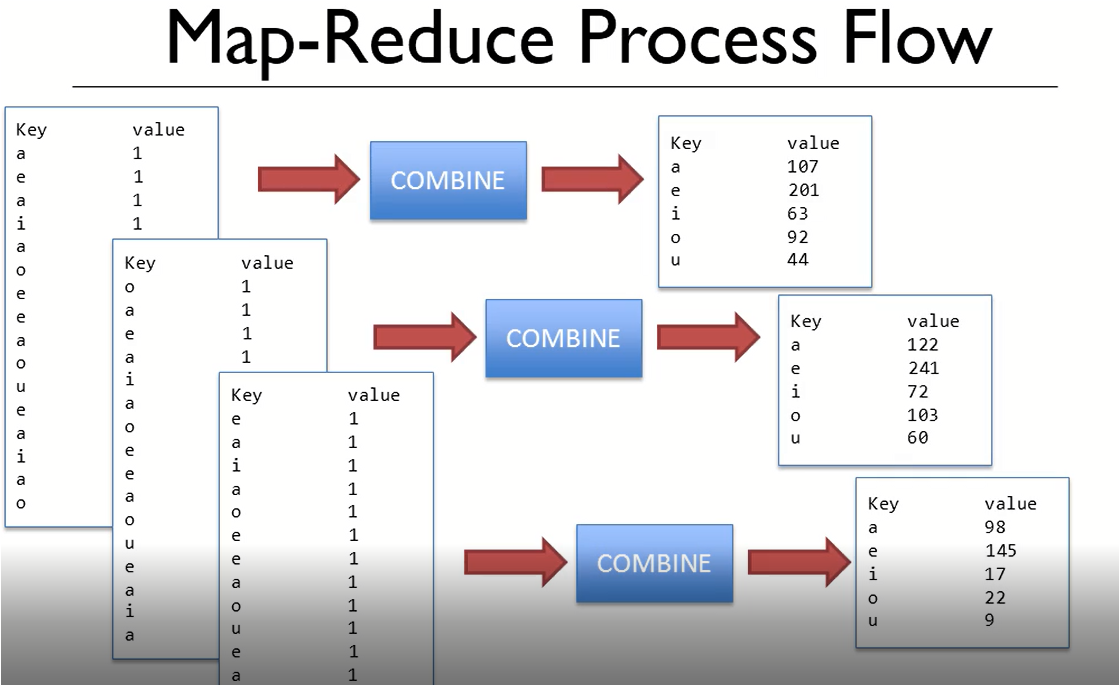


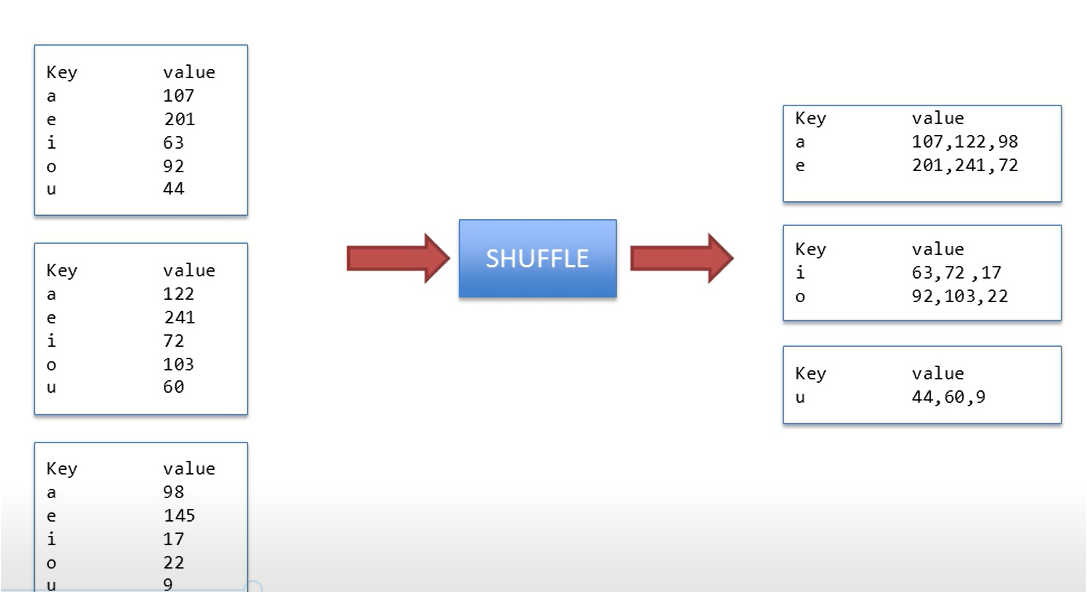
**MapReduce with Combiner Step**

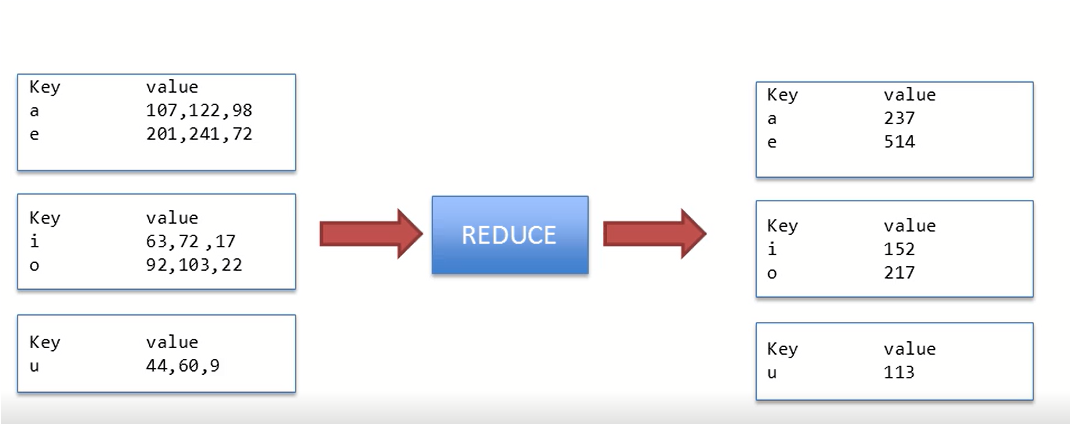


როგორც ვხედავთ COMBINE-საფეხურის გამოყენების შემთხვევაში პროგრამის მუშაობის დრო შემცირდა 30%-ით.

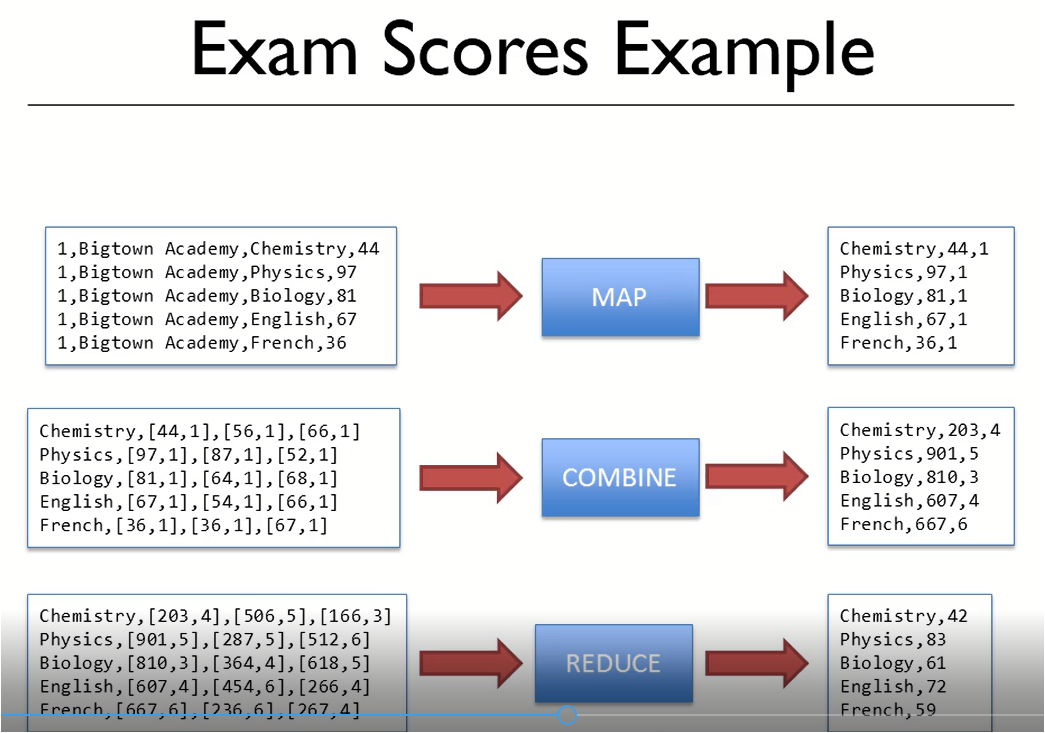
****





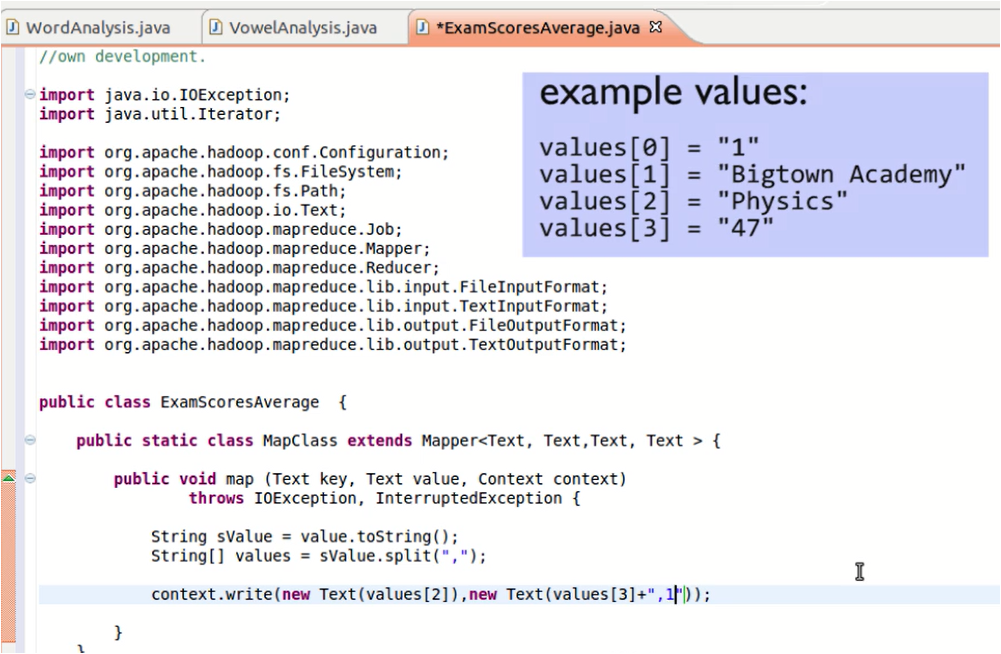


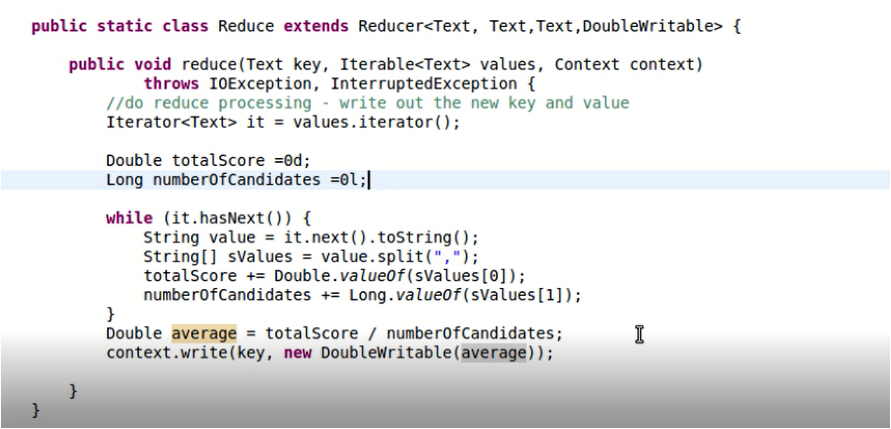
Calculate average Scores

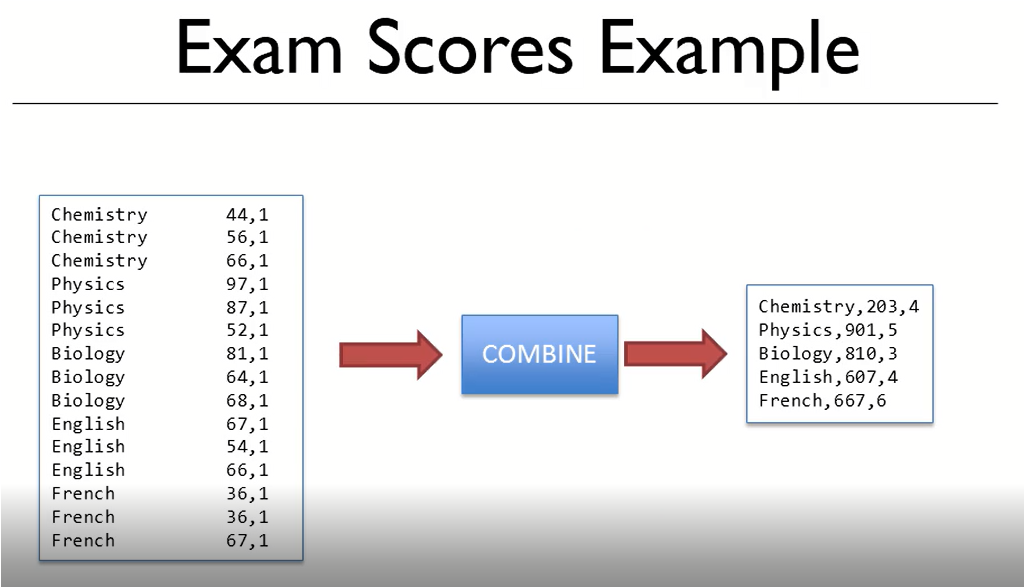


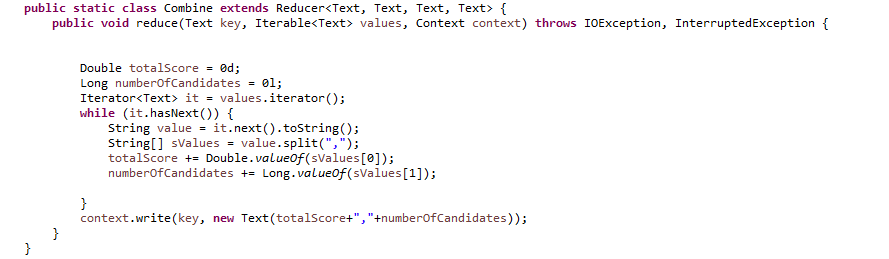
ამ ამოცანაში ჩვენ გამოსათვლელი გვაქვს საშუალო ქულა თითოეულ საგანში. ზემოთ მოყვანილია MAP, COMBINE, REDUCE პროცესების მიერ შესასრულებელი ამოცანები.

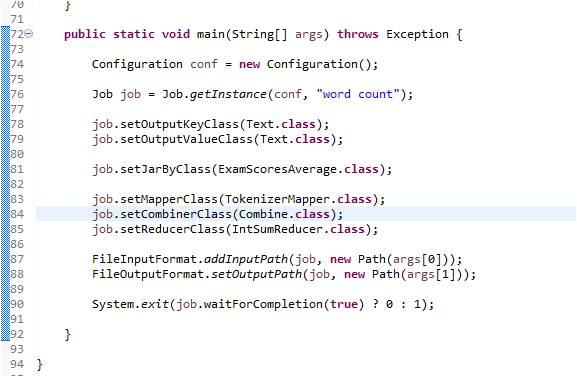
ამ შემთხვევაში values[3] წარმოადგენს ჩვენ შემთხვევაში key-ს ხოლო value-ს როლში ავიღოთ მიღებული ქულა და 1-იანი რომელიც მიუთითებს იმ სტუდენტების რაოდენობაზე რამდენმაც მიიღო ეს ქულა. ჩვენ გვექნება მაგ. Chemistry,44,1. იმ შემთხვევაში თუ აღმოჩნდებოდა რომ კიდევ გვყავს ისეთი სტუდენტი რომელმაც ქიმიაში მიიღო 44 ქულა მაშინ კიდევ ერთი ასეთი ჩანაწერი გვექნებოდა

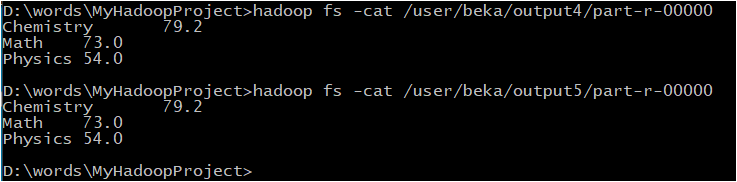












How to Chain MapReduce Job in Hadoop

**ChainMapper in Hadoop**

ChainMapper is one of the predefined MapReduce class in Hadoop. *ChainMapper class allows you to use multiple Mapper classes within a single Map task*. The Mapper classes are invoked in a chained fashion where the output of the first mapper becomes the input of the second, and so on until the last Mapper, the output of the last Mapper will be written to the task’s output.  
You can add mappers to a ChainMapper using **addMapper()** method.

**ChainReducer in Hadoop**

The ChainReducer class allows to chain multiple Mapper classes after a Reducer within the Reducer task. For each record output by the Reducer, the Mapper classes are invoked in a chained fashion. The output of the reducer becomes the input of the first mapper and output of first becomes the input of the second, and so on until the last Mapper, the output of the last Mapper will be written to the task’s output.

To add a Mapper class to the chain reducer you can use **addMapper()** method.  
To set the Reducer class to the chain job you can use **setReducer()** method.

**Chaining MapReduce job**

Using the ChainMapper and the ChainReducer classes it is possible to compose MapReduce jobs that look like **[MAP+ / REDUCE MAP\*]**.

#### Chaining MapReduce job

Using the ChainMapper and the ChainReducer classes it is possible to compose MapReduce jobs that look like **[MAP+ / REDUCE MAP\*]**.

When you are using chained MapReduce you can have a combination as follows-

1. One or more mappers
2. Single Reducer
3. Zero or more mappers (optional and to be used only if chained reducer is used)

When you are using chained MapReduce job the data from mappers or reducer is stored (and used) in the memory rather than on disk that reduces the disk IO to a large extent.

**import** java.io.IOException;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.conf.Configured;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Reducer;

**import** org.apache.hadoop.mapreduce.lib.chain.ChainMapper;

**import** org.apache.hadoop.mapreduce.lib.chain.ChainReducer;

**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

**import** org.apache.hadoop.mapreduce.lib.map.InverseMapper;

**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

**import** org.apache.hadoop.util.Tool;

**import** org.apache.hadoop.util.ToolRunner;

**public** **class** StockTrans **extends** Configured **implements** Tool{

// Mapper 1

**public** **static** **class** StockFieldMapper **extends** Mapper<LongWritable, Text, Text, IntWritable>{

**private** Text symbol = **new** Text();

**public** **void** map(LongWritable key, Text value, Context context)

**throws** IOException, InterruptedException {

// Splitting the line on tab

String[] stringArr = value.toString().split("\t");

//Setting symbol and transaction values

symbol.set(stringArr[0]);

Integer trans = Integer.*parseInt*(stringArr[2]);

context.write(symbol, **new** IntWritable(trans));

}

}

// Mapper 2

**public** **static** **class** UpperCaseMapper **extends** Mapper<Text, IntWritable, Text, IntWritable>{

**public** **void** map(Text key, IntWritable value, Context context)

**throws** IOException, InterruptedException {

String symbol = key.toString().toUpperCase();

context.write(**new** Text(symbol), value);

}

}

// Reduce function

**public** **static** **class** TotalTransReducer **extends** Reducer<Text, IntWritable, Text, IntWritable>{

**public** **void** reduce(Text key, Iterable<IntWritable> values, Context context)

**throws** IOException, InterruptedException {

**int** sum = 0;

**for** (IntWritable val : values) {

sum += val.get();

}

context.write(key, **new** IntWritable(sum));

}

}

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

**int** exitFlag = ToolRunner.*run*(**new** StockTrans(), args);

System.*exit*(exitFlag);

}

@Override

**public** **int** run(String[] args) **throws** Exception {

Configuration conf = **new** Configuration();

Job job = Job.*getInstance*(conf, "Stock transactio");

job.setJarByClass(getClass());

// MapReduce chaining

Configuration map1Conf = **new** Configuration(**false**);

ChainMapper.*addMapper*(job, StockFieldMapper.**class**, LongWritable.**class**, Text.**class**,

Text.**class**, IntWritable.**class**, map1Conf);

Configuration map2Conf = **new** Configuration(**false**);

ChainMapper.*addMapper*(job, UpperCaseMapper.**class**, Text.**class**, IntWritable.**class**,

Text.**class**, IntWritable.**class**, map2Conf);

Configuration reduceConf = **new** Configuration(**false**);

ChainReducer.*setReducer*(job, TotalTransReducer.**class**, Text.**class**, IntWritable.**class**,

Text.**class**, IntWritable.**class**, reduceConf);

ChainReducer.*addMapper*(job, InverseMapper.**class**, Text.**class**, IntWritable.**class**,

IntWritable.**class**, Text.**class**, **null**);

job.setOutputKeyClass(IntWritable.**class**);

job.setOutputValueClass(Text.**class**);

FileInputFormat.*addInputPath*(job, **new** Path(args[0]));

FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

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

}

}

ახალი პროგრამის გამშვები:

