Lecture 07

Map Reduce API-s

Zoran B. Djordjević

Reference

- These slides follow the text of the book:

  Hadoop in Action, by Chuck Lam, Manning 2011

New vs. Old Map Reduce API

- Until release 0.183 Hadoop supported a particular Map Reduce API. With release 0.20 a new API is introduced which simplified coding and added some new features.
- It appears that many books and examples on the Internet are written in the old API. In order to help you use those examples we will present some MapReduce classes in both API-s and describe key differences. Differences are not huge and are easy to reconcile.
- In the end you will end up using templates in either API and pay attention to little else but details in methods `map()` and `reduce()`.

Source of Data, National Bureau of Economic Research

- http://www.nber.org/data/
- http://data.nber.org/patents/
- Download acite75_99.zip (82MB) and apat63_99.zip (56Mb)

Description	Documentation	Data -- Pkzipped	
		SAS .tpt	ASCII CSV
Overview	overview.txt	--	
Pairwise citations data	Cite75_99.txt	Cite75_99.zip -- (68 Mb)	acite75_99.zip -- (82 Mb)
Patent data, including constructed variables	pat63_99.txt	pat63_99.zip -- (90Mb)	apat63_99.zip -- (56Mb)
Assignee names	coname.txt	coname.zip -- (2Mb)	aconame.zip -- (2Mb)
Contains the match to CUSIP numbers	match.txt	match.zip -- (130Kb)	amatch.zip -- (98Kb)
Individual inventor records	inventor.txt	inventor.zip -- (98Mb)	ainventor.zip -- (82Mb)
Class codes with corresponding class names	classes.txt	--	
Country codes with corresponding country names	countries.txt		
Class, technological category, and technological	class_match.txt		

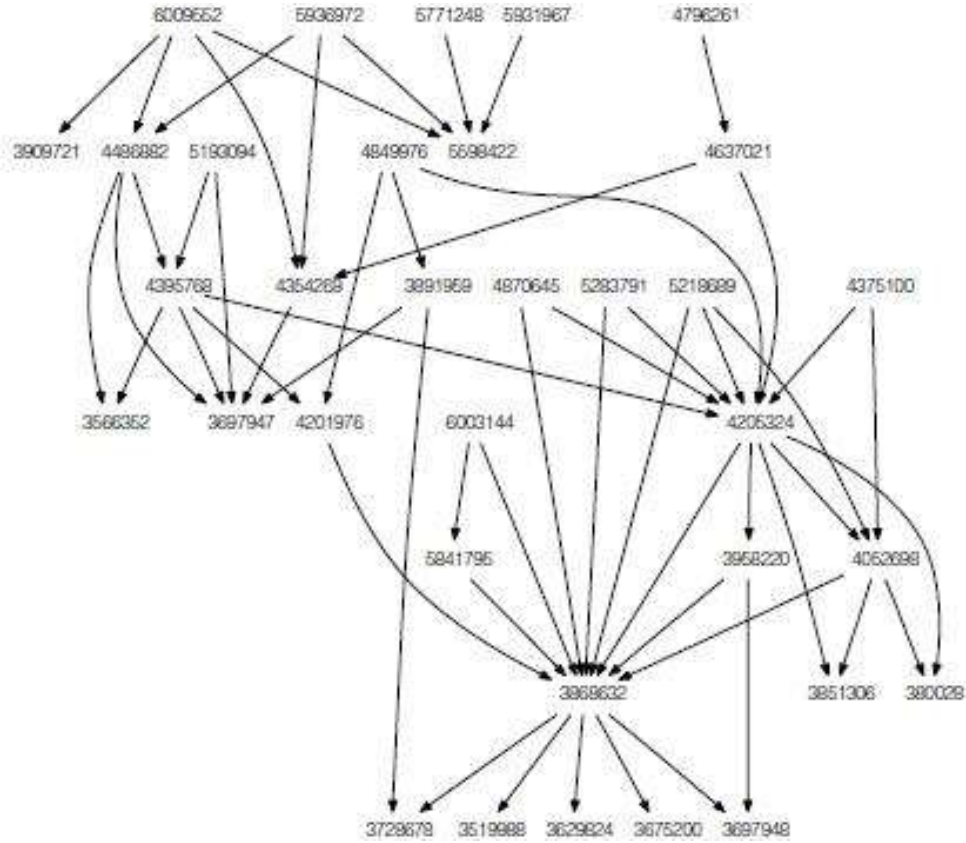
Patent Citation Data, cite75_99.txt File

- Source: US Patent office
- This file includes all US patent citations for utility patents granted in the period 1-Jan-75 to 31-Dec-99.
- No. of observations: 16,522,438
- Variable Name variable type Characters Contents
- CITING numeric 7 Citing Patent Number
- CITED numeric 7 Cited Patent Number
- The file is sorted by Citing Patent Number.

"CITING", "CITED"
6009552,5278871
6009552,5598422
6009553,4131849
6009553,4517669
6009553,4519068
6009553,4590473
6009553,4636791
6009553,5671255
.....

Patent Citation Data

- If you're only reading the data file, the citation data appears to be a bunch of numbers.
- One way is to visualize it as a graph. Bellow we show a portion of this citation graph .
- Some patents are cited often whereas others aren't cited at all. Patents like 5936972 and 6009552 cite a similar set of patents (4354269, 4486882, 5598422) , though they don't cite each other.
- We could use Hadoop to derive descriptive statistics about this patent data, and look for interesting, non-obvious, patterns.



Patent Description Data, pat63_99.txt file

- The other data set we could use is the patent description data. It has the patent number, the patent application year, the patent grant year, the number of claims, and other metadata about patents.

Source: USPTO, and Jaffe and Trajtenberg computations.
The Pat63_99 file includes all utility patents in the USPTO's TAF database granted during the period 1963 to December 1999. Classification information reflects the U.S. Patent Classification System as of December 31, 1999.

No. of observations: 2,923,922

USPTO Original Variables:

Variable Name	Variable type	Characters	Contents
patent	numeric	7	Patent Number
gyear	numeric	12	Grant Year
gdate	numeric	12	Grant Date
appyear	numeric	12	Application Year
country	character	3	Country of First Inventor
postate	character	3	State of First Inventor (if US)
assignee	numeric	12	Assignee Identifier (missing 1963-1967)
asscode	numeric	12	Assignee Type (see below)
claims	numeric	12	number of Claims
Nclass	numeric	12	Main Patent Class (3 digit)

New Variables:

Variable Name	Variable type	Characters	Contents
cat	numeric	12	Technological Category
subcat	numeric	12	Technological Sub-Category

apat63_99.txt

C:\VMs\sharedfolder\patents>head -40 apat63_99.txt

```
"PATENT","GYEAR","GDATE","APPYEAR","COUNTRY","POSTATE","ASSIGNEE","ASSCODE","CLAIMS","NCLASS","
CAT","SUBCAT","CMADE","CRECEIVE","RATIOCIT","GENERAL","ORIGINAL","FWDAPLAG","BCKGTLAG","SELFCTU
B","SELFCTLB","SECDUPBD","SECDLWBD"
3070801,1963,1096,,"BE","",,1,,269,6,69,,1,,0,,,,,,,
3070802,1963,1096,,"US","TX",,1,,2,6,63,,0,,,,,,,,,
3070803,1963,1096,,"US","IL",,1,,2,6,63,,9,,0.3704,,,,,,,
3070804,1963,1096,,"US","OH",,1,,2,6,63,,3,,0.6667,,,,,,,
3070805,1963,1096,,"US","CA",,1,,2,6,63,,1,,0,,,,,,,
3070806,1963,1096,,"US","PA",,1,,2,6,63,,0,,,,,,,,,
3070807,1963,1096,,"US","OH",,1,,623,3,39,,3,,0.4444,,,,,,,
3070808,1963,1096,,"US","IA",,1,,623,3,39,,4,,0.375,,,,,,,
3070809,1963,1096,,"US","AZ",,1,,4,6,65,,0,,,,,,,,,
3070810,1963,1096,,"US","IL",,1,,4,6,65,,3,,0.4444,,,,,,,
3070811,1963,1096,,"US","CA",,1,,4,6,65,,8,,0,,,,,,,
3070812,1963,1096,,"US","LA",,1,,4,6,65,,3,,0.4444,,,,,,,
3070813,1963,1096,,"US","NY",,1,,5,6,65,,2,,0,,,,,,,
3070814,1963,1096,,"US","MN",,2,,267,5,59,,2,,0.5,,,,,,,
3070815,1963,1096,,"US","CO",,1,,7,5,59,,1,,0,,,,,,,
3070816,1963,1096,,"US","OK",,1,,114,5,55,,4,,0,,,,,,,
3070817,1963,1096,,"US","RI",,2,,114,5,55,,5,,0.64,,,,,,,
3070818,1963,1096,,"US","IN",,1,,441,6,69,,4,,0.625,,,,,,,
3070819,1963,1096,,"US","TN",,4,,12,6,63,,0,,,,,,,,,
3070820,1963,1096,,"GB","",,2,,12,6,63,,0,,,,,,,,,
3070821,1963,1096,,"US","IL",,2,,15,6,69,,1,,0,,,,,,,
3070822,1963,1096,,"US","NY",,2,,401,1,12,,4,,0.375,,,,,,,
3070823,1963,1096,,"US","MI",,1,,401,1,12,,8,,0.6563,,,,,,,
3070824,1963,1096,,"US","IL",,1,,401,1,12,,5,,0.48,,,,,,,
3070825,1963,1096,,"US","IL",,1,,401,1,12,,7,,0.6531,,,,,,,
3070826,1963,1096,,"US","IA",,1,,401,1,12,,1,,0,,,,,,,
3070827,1963,1096,,"US","CA",,4,,401,1,12,,2,,0.5,,,,,,,
3070828,1963,1096,,"US","CT",,2,,16,5,59,,4,,0.625,,,,,,,
3070829,1963,1096,,"FR","",,3,,16,5,59,,5,,0.48,,,,,,,
3070830,1963,1096,,"US","NH",,2,,16,5,59,,0,,,,,,,,,
```

Definition of some attributes of Patent Data Set

Attribute	Description
PATENT	Patent number
GYEAR	Grant year
GDATA	Grant date, given as the number of days elapsed since January 1, 1960
APPYEAR	Application year (available only for patents granted since 1967)
COUNTRY	Country of first inventor
POSTATE	State of first inventory (if country is U.S.)
ASSIGNEE	Numeric identifier for assignee (i.e., patent owner)
ASSCODE	One-digit (1-9) assignee type. (The assignee type includes U.S. individual, U.S. government, U.S. organization, non-U.S. individual, etc.)
CLAIMS	Number of claims (available only for patents granted since 1975)
NCLASS	3-digit main patent class

List patents and patents that cite them

- We want to invert citation index. Rather than listing `citing` vs. `cited`, we want to invert the data and list `cited` vs. all `citing`. We expect to generate a file that would look like:

```
"CITED"  "CITING"
1000033 4190903,4975983
1000043 4091523
1000044 4082383,4055371
1000045 4290571
1000046 5918892,5525001
1000067 5312208,4944640,5071294
```

- Patent `1000067` is cited by patents `5312208,4944640` and `5071294`

- We will accomplish this objective by writing a MapReduce class `Inverter.java`.
- That class will serve as a template for our future developments.
- In future classes, most of the lines of code will remain the same, while we will change a few lines in the `map()` and `reduce()` methods.

Inverter.java, Old API, Template Class

```
package edu.hu.bigdata;
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

Inverter.java

```
public class Inverter extends Configured implements Tool {
    public static class MapClass extends MapReduceBase
        implements Mapper<Text, Text, Text, Text> {
        public void map(Text key, Text value,
                        OutputCollector<Text, Text> output,
                        Reporter reporter) throws IOException {
            output.collect(value, key);
        }
    }
    public static class Reduce extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterator<Text> values,
                           OutputCollector<Text, Text> output,
                           Reporter reporter) throws IOException {
            String csv = "";
            while (values.hasNext()) {
                if (csv.length() > 0) csv += ",";
                csv += values.next().toString();
            }
            output.collect(key, new Text(csv));
        }
    }
```

Inverter.java

```
public int run(String[] args) throws Exception {
        Configuration conf = getConf();
        JobConf job = new JobConf(conf, Inverter.class);
        Path in = new Path(args[0]);
        Path out = new Path(args[1]);
        FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);
        job.setJobName("Inverter");
        job.setMapperClass(MapClass.class);
        job.setReducerClass(Reduce.class);
        job.setInputFormat(KeyValueTextInputFormat.class);
        job.setOutputFormat(TextOutputFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.set("key.value.separator.in.input.line", ",");
        JobClient.runJob(job);
        return 0;
    }
    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(),new
Inverter(),args);
        System.exit(res);
    }
}
```

Result of Running `Inverter` job

- We compiled `Inverter.java` in Eclipse with Build Path containing:
  - `hadoop-core-2.0.0-mr1-cdh4.6.0.jar`,
  - `hadoop-common-2.0.0-chd4.6.0.jar` and `commons-cli-1.2.jar`
- Exported `invertercounter.jar`, moved `cite-99.txt` to HDFS `input` directory on MRv1 VM and ran command:

```
$ hadoop jar inverter.jar edu.hu.bigdata.Inverter input outputinv
```

- Command `hadoop fs –tail outputinv/part-00000` gave the following:

```
......
999936  5014973,5642878
999940  4022472,3876070
999941  5447466
999945  5569166,5207231
999949  5640640
999951  5374468,5316622
999957  5755359
999961  5738381,5782495,5878901,4871140,4832301,5048788,4171117,4262874
999965  5052613
999968  3916735
999971  3965843
999972  4038129
999973  5427610,4900344
999974  4728158,4560073,5464105
. . . . .
```

Programming Convention

- Standard convention is that a single class, `Inverter` in this case, completely defines each MapReduce job.
- Hadoop requires the `Mapper` and the `Reducer` to be their own static classes. These classes are quite small, and our template includes them as inner classes in the `Inverter` class.
- The advantage is that everything fits in one file, simplifying code management.
- These inner classes are independent and don't interact much with the `Inverter` class.
- Various nodes with different JVMs clone and run the Mapper and the Reducer during job execution , whereas the rest of the job class is executed *only* at the client machine.

`Inverter` **`extends`** `Configured` **`implements`** `Tool`

- `Inverter is` a subclass of `Configured`, which is an implementation of the `Configurable` interface.
- All implementations of `Tool` need to implement `Configurable` (since `Tool` extends it).
- Extending, i.e. subclassing, `Configured` is a way to achieve this.
- The `run()` method obtains the `Configuration` using `getConf()` method of `Configurable` interface

`org.apache.hadoop.conf.Configured` class

```
@InterfaceAudience.Public
@InterfaceStability.Stable
public class Configured
extends Object
implements Configurable
```

`Configured` is the base class for objects that may be configured with a `Configuration`

Constructors:

`Configured()` Constructs a `Configured` object.

`Configured(Configuration conf)` Construct a `Configured` object based on specified `Configuration` object `conf`.

Methods:

`Configuration getConf()` Return the configuration used by this object.

`void setConf(Configuration conf)` Set the `Configuration` by passing a `Configuration` object `conf`.

`org.apache.hadoop.util.Tool` class

```
@InterfaceAudience.Public
@InterfaceStability.Stable
public interface Tool extends Configurable
```

`Tool` interface supports handling of generic command-line options.

`Tool`, is the standard for any MapReduce tool/application.

The tool/application should delegate the handling of standard command-line options to `ToolRunner.run(Tool, String[])` and only handle its custom arguments.

Method Summary

```
int run(String[] args)
```

Execute the command with the given arguments.

`Tool` has methods inherited from interface `org.apache.hadoop.conf.Configurable`

`getConf(), setConf()`

- A typical implementation of the `Tool` interface looks like `Inverter` class

org.apache.hadoop.util.Tool class

```
public class Inverter extends Configured implements Tool {
    public int run(String[] args) throws Exception {  // must implement run()
        // Configuration processed by ToolRunner
        Configuration conf = getConf();
        // Create a JobConf using the provided conf
        JobConf job = new JobConf(conf, MyApp.class);
        // Process custom command-line options
        Path in = new Path(args[1]);
        Path out = new Path(args[2]);
        FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);
        // Specify various job-specific parameters
        job.setJobName("inverter");
        job.setInputPath(in);
        job.setOutputPath(out);
        job.setMapperClass(MapClass.class);
        job.setReducerClass(Reducer.class);
        // Submit the job, then poll for progress until the job is complete
        JobClient.runJob(job);
        return 0;
    }
    public static void main(String[] args) throws Exception {
        // Let ToolRunner handle generic command-line options
        int res = ToolRunner.run(new Configuration(), new Inverter(), args);
        System.exit(res);
    }
}
```

org.apache.hadoop.util.ToolRunner

```
@InterfaceAudience.Public
@InterfaceStability.Stable
public class ToolRunner extends Object
```

- A utility to help run `Tools`.
- `ToolRunner` can be used to run classes implementing `Tool` interface. It works in conjunction with `GenericOptionsParser` to parse the generic command line arguments of command `hadoop` and modifies the `Configuration` of the `Tool`. The application-specific options are passed along without being modified.
- Constructor Summary

`ToolRunner()`

- Method Summary

  `static boolean confirmPrompt(String prompt)`
- Print out a prompt to the user, and return true if the user responds with "y" or "yes".

  `static void printGenericCommandUsage(PrintStream out)`
- Prints generic command-line argurments and usage information.

  `static int run(Configuration conf, Tool tool, String[] args)`
- Runs the given `Tool by Tool.run(String[])`, after parsing with the given generic arguments.

  `static int run(Tool tool, String[] args)`
- Runs the `Tool` with its `Configuration`.

“Client” portion of `Inverter` class

- The core of `Inverter` class is within the `run()` method.
- `run()` method is also known as the *driver*.
- The driver instantiates, configures, and passes a `JobConf` object named `job` to `JobClient. runJob()` to start the MapReduce job.
- The `JobClient` class, in turn, will communicate with the `JobTracker` to start the job across the cluster.
- The `JobConf` object holds all configuration parameters necessary for the job to run.
- `JobConf` is fed configuration parameters from the `Configuration` object which is retrieved by the `getConf()` method of `Configurable` interface our main class implements.

“Client” portion of Inverter

- The driver needs to specify the basic parameters for every `job`
  - `input Path`,
  - `output Path`,
  - `MapperClass`, and
  - `ReducerClass`.
- In addition, each `job` can reset the default properties, such as
  - `InputFormat`,
  - `OutputFormat`
- One can also call the `set()` method on the `JobConf` object to set up any configuration parameter.
- Once you pass the `JobConf` object to `JobClient.runJob()`, it’s treated as the master plan for the job. It becomes the blueprint for how the `job` will be run.

org.apache.hadoop.mapred.JobConf

```
@InterfaceAudience.Public
@InterfaceStability.Stable
public class JobConf extends Configuration
```

- A map/reduce job configuration.
- `JobConf` is the primary interface for a user to describe a map-reduce job to the Hadoop framework for execution. The framework tries to execute the job as-is described by JobConf,
- Some configuration parameters might have been marked as final by administrators and hence cannot be altered.
- While some job parameters are straight-forward to set (e.g. `setNumReduceTasks(int)`), some parameters interact subtly with the rest of the framework and/or job-configuration and are relatively more complex for the user to control finely (e.g. `setNumMapTasks(int)`).
- `JobConf` typically specifies the `Mapper`, combiner (if any), `Partitioner`, `Reducer`, `InputFormat` and `OutputFormat` implementations to be used etc.
- Optionally `JobConf` is used to specify other advanced facets of the job such as `Comparators` to be used, files to be put in the `DistributedCache`, whether or not intermediate and/or job outputs are to be compressed (and how), debugability via user-provided scripts (`setMapDebugScript(String)/setReduceDebugScript(String)`), for doing post-processing on task logs, task's `stdout`, `stderr`, `syslog`. and etc.

org.apache.hadoop.mapred.JobConf

- The following demonstrates how to configure a job via `JobConf`:

```
// Create a new JobConf
JobConf job = new JobConf(new Configuration(),Inverter.class);

// Specify various job-specific parameters
job.setJobName("Inverter");

FileInputFormat.setInputPaths(job, new Path("in"));
FileOutputFormat.setOutputPath(job, new Path("out"));

job.setMapperClass(Inverter.MapClass.class);
job.setCombinerClass(Inverter.Reducer.class);
job.setReducerClass(Inverter.Reducer.class);

job.setInputFormat(SequenceFileInputFormat.class);
job.setOutputFormat(SequenceFileOutputFormat.class);
```

JobConf and Configuration

- The `JobConf` object has many parameters, but we don't want to program the driver to set up all of them.
- The configuration files of the Hadoop installation are a good starting point.
- When starting a job from the command line, the user may also want to pass extra arguments to alter the job configuration.
- The driver can define its own set of commands and process the user arguments itself to enable the user to modify some of the configuration parameters.
- Configuration files are represented by the `Configuration` object.
- `ToolRunner` internally runs `GenericOptionsParser` object which reads and parses the command line arguments.

`org.apache.hadoop.conf.Configuration` class

```
@InterfaceAudience.Public
@InterfaceStability.Stable
public class Configuration
extends Object
implements Iterable<Map.Entry<String,String>>, Writable
```

Provides access to configuration parameters.

Resources

Configurations are specified by resources. A resource contains a set of name/value pairs as XML data. Each resource is named by either a String or by a Path. If named by a String, then the classpath is examined for a file with that name. If named by a Path, then the local file system is examined directly, without referring to the classpath. Unless explicitly turned off, Hadoop by default specifies two resources, loaded in-order from the classpath:

`core-default.xml` : Read-only defaults for hadoop

`core-site.xml`: Site-specific configuration for a given hadoop installation.

Applications may add additional resources. On our MRv1 VM, file `core-site.xml` resides in the directory `/etc/hadoop/conf.pseudo.mr1`

`org.apache.hadoop.conf.Configuration` class

`Final Parameters`

- On our MRv1 VM, file `core-site.xml` resides in the directory `/etc/hadoop/conf.pseudo.mr1`
- Configuration parameters may be declared *final*. Once a resource declares a value final, no subsequently-loaded resource can alter that value.
- For example, one might define a final parameter with:

```
<property>
  <name>dfs.hosts.include</name>
  <value>/etc/hadoop/conf/hosts.include</value>
  <final>true</final>
</property>
```

`org.apache.hadoop.conf.Configuration` class

Variable Expansion

- Value strings are first processed for *variable expansion*.
- The available properties are:
  - other properties defined in this configuration file; and,
  - if a name is undefined there, properties in `System.getProperties()`
- For example, if a configuration resource contains the following property definitions :

```
<property>
  <name>basedir</name>
  <value>/user/${user.name}</value>
</property>
<property>
  <name>tempdir</name>
  <value>${basedir}/tmp</value>
</property>
```

- When `conf.get("tempdir")` is called, then `${basedir}` will be resolved to another property in this configuration, while `${user.name}` would ordinarily be resolved to the value of the System property with that name.

GenericOptionsParser, Tool and ToolRunner

- Hadoop comes with a few helper classes for making it easier to run jobs from the command line.
- `GenericOptionsParser` is a class that interprets common Hadoop command-line options and sets them on a `Configuration` object for your application to use as desired.
- You don't usually use `GenericOptionsParser` directly. It is more convenient to implement the `Tool` interface and run your application with the `ToolRunner`, which uses `GenericOptionsParser` internally:

```
public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(
            new Configuration(), new Inverter(), args);
    System.exit(res);
}
```

org.apache.hadoop.util.GenericOptionsParser

http://hadoop.apache.org/docs/stable/api/org/apache/hadoop/util/GenericOptionsParser.html

- `GenericOptionsParser` is a utility to parse command line arguments generic to the Hadoop framework. `GenericOptionsParser` recognizes several standard command line arguments, enabling applications to easily specify a `namenode`, a `jobtracker`, and additional configuration resources.

Generic Options

- The supported generic options are:

`-conf <configuration file>`	specify a configuration file
`-D <property=value>`	user value for given property
`-fs <local|namenode:port>`	specify a namenode
`-jt <local|jobtracker:port>`	specify a job tracker
`-files` <comma separated list of files>	specify comma separated files to be copied to the map reduce cluster
`-libjars <comma separated list of jars>`	specify comma separated jar files to include in the classpath.
`-archives <comma separated list of archives>`	specify comma separated archives to be unarchived on the compute machines.

- Generic command line arguments **might** modify `Configuration` objects, given to constructors.
- The functionality is implemented using Commons CLI.

Example of command line option use

- For example, we would normally execute the `Inverter` class using a command line like:

```
$ hadoop jar inverter.jar edu.hu.bd.Inverter input output
```

- Had we wanted to run the job only to see the mapper's output (which you may want to do for debugging purposes), we could set the number of reducers to zero with the option `-D mapred.reduce.tasks=0`.

```
$ hadoop jar inverter.jar edu.hu.bd.Inverter
      -D mapred.reduce.tasks=0   input output
```

- This works even though our program doesn't explicitly understand the `-D` option.
- By using `ToolRunner`, `Inverter` will automatically support the options understood by `GenericOptionsParser`

Mapper and Reducer

- The convention for our template is to call the Mapper class `MapClass` and the Reducer class `Reduce`.
- The naming would seem more symmetric had we called the `Mapper` class `Map`. However, Java already has a class (interface) named `Map`.
- Both the Mapper and the Reducer extend `MapReduceBase`, which is a small class providing no-op implementations to the `configure()` and `close()` methods required by the two interfaces.
- The `configure()` and `close()` methods are life cycle methods and one could use them to set up and clean up the map (reduce) tasks.
- Usually you do not need to override `configure()` and `close()` except for more advanced jobs.

Signature of Mapper and Reducer classes

- The signatures for the Mapper class and the Reducer class are:

```
public static class MapClass extends MapReduceBase
       implements Mapper<K1, V1, K2, V2> {
       public void map(K1 key, V1 value,
                       OutputCollector<K2, V2> output,
                        Reporter reporter)
                  throws IOException { }
}
public static class Reduce extends MapReduceBase
       implements Reducer<K2, V2, K3, V3> {
       public void reduce(K2 key, Iterator<V2> values,
                          OutputCollector<K3, V3> output,
                          Reporter reporter)
                  throws IOException { }
}
```

`map()` and `reduce()` methods

- The center of action for the Mapper class is the `map()` method and for the Reducer class the `reduce()` method.
- Each invocation of the `map()` method is given a key/value pair of types K1 and V1, respectively.
- The key/value pairs generated by the mapper and reducer are sent out via the `collect()` method of the `OutputCollector` object.
- Somewhere in your map() method you need to call

```
output.collect((K2) k, (V2) v);
```

- Each invocation of the `reduce()` method at the reducer is given a key of type K2 and a list of values of type V2.
- Note that those are the same K2 and V2 types used in the Mapper. The `reduce()` method loops through all the values of type V2.

```
while (values.hasNext()) { V2 v = values.next();  . . . }
```

- Somewhere in the `reduce()` method you'll call

```
output.collect((K3) k, (V3) v);
```

to send results out

org.apache.hadoop.mapred.OutputCollector

```
@InterfaceAudience.Public
@InterfaceStability.Stable
public interface OutputCollector<K,V>
```

- Collects the `<key, value>` pairs output by `Mappers` and `Reducers`.
- `OutputCollector` is the generalization of the facility provided by the MapReduce framework to collect data output by either the `Mapper` or the `Reducer` i.e. intermediate outputs or the final output of the job.

- Method Summary

```
void collect(K key, V value) throws IOExceptionAdds
```

Adds a key/value pair to the output.

Type matching

- In addition to having consistent `K2` and `V2` types across Mapper and Reducer, you'll also need to ensure that the key and value types used in Mapper and Reducer are consistent with the input format, output key class, and output value class set in the driver.
- The use of `KeyValueTextInputFormat` in the `Inverter` means that `K1` and `V1` must both be type `Text`.
- The driver must call `setOutputKeyClass()` and `setOutputValueClass()` with the classes of `K2` and `V2`, respectively.
- All the key and value types must be subtypes of `Writable`, which means they are `Serializable` and serialization interface of Hadoop could send them around the distributed cluster.
- In fact, the key types implement `WritableComparable`, a subinterface of `Writable`. The key types need to additionally support the `compareTo()` method, as keys need to be sorted in various places in the MapReduce framework.

Counting Citations (Citings)

- Much of what we think of as statistics is counting, and many basic Hadoop jobs involve counting.
- For the patent citation data, we may want the number of citations a patent has received. The desired output would look like this:

```
1000006 1
1000007 3
1000011 7
1000017 1
```

- In each record, a patent number is associated with the number of citations it has received. We will write a MapReduce program for this task.
- We hardly ever write a MapReduce program from scratch. You have an existing MapReduce program `Inverter` that processes the data in a similar way. We copy and rename class `Inverter` into `InverterCounter` and modify it until it does what we want .

InverterCounter, Modify Reducer

- We can modify Inverter to output the count instead of the list of citing patents. We need the modifications only at the `Reducer`.
- If we choose to output the count as an `IntWritable`, we need to specify `IntWritable` in three places in the `Reducer` code. We called them V3 in our previous notation.

```
public static class Reduce extends MapReduceBase implements
        Reducer<Text, Text, Text, IntWritable> {
        public void reduce(Text key, Iterator<Text> values,
        OutputCollector<Text, IntWritable> output,
        Reporter reporter) throws IOException {
        int count = 0;
                while (values.hasNext()) {
                 values.next(); count++;
        }
       output.collect(key, new IntWritable(count)); } }
```

- By changing a few lines and matching class types, we created a new MapReduce program.

Running `InverterCounter`

- We compile `InverterCounter` very much like class `Inverter`.
- We run it using the same input file and send output to a new directory:

```
$ hadoop jar inverter.jar edu.hu.bigdata.InverterCounter
input outputcount
```

`$ hadoop fs –tail outputcount/part-00000` gives

```
...
999945 2
999949 1
999951 2
999957 1
999961 9
999965 1
999968 1
999971 1
999972 1
999973 2
999974 3
...
```

Build Histogram of Citations

- Now that we know that many patents are cited once, twice and so forth, we would like to know the exact number of patents cited once, cited twice, and so forth.
- This type of result we will call the Histogram of the Citation Counts.
- We expect a large number of patents to have been cited once, and a small number may have been cited hundreds of times.
- Initially we will use the result produced by `InverterCounter` as the input into new MapReduce program.

Data Flow

- The first step to writing a new MapReduce program is to figure out the data flow. We start with the data produced by `InverterCounter`:

```
1000006 1
1000007 3
1000011 7
1000017 1
```

- We want to know how may times a number (`citation_count`) , for example 3, appears in the entire file and how many times number 7 appears, and so on.
- Our new mapper will read a record and ignore the patent number or rather replace it with number 1.
- The Mapper will output an intermediate key/value pair of `<citation_count, 1>` .
- The Reducer will sum up the number of 1s for each `citation_count` and output the total.

Data Types

- After figuring out the data flow, we need to decide on the types for the key/value pairs—`K1, V1, K2, V2, K3,` and `V3` for the input, intermediate, and output key/value pairs.
- We will keep using the `KeyValueTextInputFormat`, which automatically breaks each input record into key/value pairs based on a separator character.
- The input format produces `K1` and `V1` as `Text` . We choose to use `IntWritable` for `K2, V2, K3,` and `V3` because we know those data must be integers and it's more efficient to use `IntWritable` than `Text`.
- We will call our new program `CitationHistogram`. Its complete listing is given on the following slides.
- We do not plan to generate a graphical representation of that histogram using Map Reduce techniques. ☺

CitationHistogram.java

```
package edu.hu.bgd;
import java.io.IOException;
import java.util.Iterator;
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.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

CitationHistogram.java

```
public class CitationHistogram extends Configured implements Tool {
    public static class MapClass extends MapReduceBase
        implements Mapper<Text, Text, IntWritable, IntWritable> {
         private final static IntWritable uno = new IntWritable(1);
         private IntWritable citationCount = new IntWritable();
         public void map(Text key, Text value,
                         OutputCollector<IntWritable, IntWritable> output,
                         Reporter reporter) throws IOException {

            citationCount.set(Integer.parseInt(value.toString()));
            output.collect(citationCount, uno);
        }
    }
    public static class Reduce extends MapReduceBase
        implements Reducer<IntWritable,IntWritable,IntWritable,IntWritable>    {
        public void reduce(IntWritable key, Iterator<IntWritable> values,
                           OutputCollector<IntWritable, IntWritable>output,
                           Reporter reporter) throws IOException {
            int count = 0;
            while (values.hasNext()) {
                count += values.next().get();
            }
            output.collect(key, new IntWritable(count));
        }
    }
```

CitationHistogram.java

```
    public int run(String[] args) throws Exception {
        Configuration conf = getConf();
        JobConf job = new JobConf(conf, CitationHistogram.class);
        Path in = new Path(args[0]);
        Path out = new Path(args[1]);
        FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);
        job.setJobName("CitationHistogram");
        job.setMapperClass(MapClass.class);
        job.setReducerClass(Reduce.class);
        job.setInputFormat(KeyValueTextInputFormat.class);
        job.setOutputFormat(TextOutputFormat.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);
        JobClient.runJob(job);
        return 0;
    }
    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(),
                                 new CitationHistogram(), args);
        System.exit(res);
    }
}
```

CitationHistogram

- The class name is now `CitationHistogram`; all references to `InverterCounter` were changed to reflect the new name.
- The `main()` method is almost always the same. The driver is mostly intact.
- The input format and output format are still `KeyValueTextInputFormat` and `TextOutputFormat`, respectively.
- The main change is that the output key class and the output value class are now `IntWritable`, to reflect the new type for `K2` and `V2`.
- We've also removed this line:
  `job.set("key.value.separator.in.input.line", ",");`
- It was setting the separator character used by `KeyValueTextInputFormat` on which it was break each input line into a key/value pair.
- In classes `Inverter` and `InverterCounter` we needed a comma for processing the original citation data. If not set, this property defaults to the tab character, which is appropriate for the citation count data.

Mapper in `CitationHistogram`

- The data flow for the mapper is similar to that of the previous mappers, only here we've chosen to define and use two new class variables: `citationCount` and `uno`.
- The `map()` method has one extra line for setting `citationCount`, which is for type casting.
- The reason for defining `citationCount` and `uno` in the class rather than inside the method is purely one of efficiency. The `map()` method will be called as many times as there are records processed at every machine.
- Reducing the number of objects created inside the `map()` method can increase performance and reduce garbage collection .
- We pass `citationCount` and `uno` to `output.collect()`

Reducer in `CitationHistogram`

- The reducer sums up the values for each key. It seems inefficient because we know all values are 1-s (`uno`, to be exact).
- Unlike in `MapClass`, the call to `output.collect()` in `Reduce` instantiates a new `IntWritable` rather than reuse an existing one.

```
output.collect(key, new IntWritable(count));
```

- This is bad programming. We could improve the performance of our program by using an `IntWritable` class variable.
- The number of times `reduce()` is called in this particular program is small, probably no more than a few thousand times. So we don't have much need to optimize this particular code, but should not be so cavalier in the future.
- We compile `CitationHistogram` using the same Build Path as previously, and export the class to `histogram.jar`

Running `CitationHistogram`

- Before we run `CitationHistogram` let us clean the directory `outputcount` where we deposited the result of `InverterCounter` job.

```
$ hadoop fs –rm –R outputcount/_*
```

- The above leaves only file `part-00000` in the directory. Next we type:

```
$  hadoop jar histogram.jar
edu.hu.bigdata.CitationHistogram outputcount outputhis
```

- As input we use `outputcount` which contains the result of `InverterCounter` job . We send results to HDFS directory `outputhis`.

- Command

```
$ hadoop fs –tail
      outputhis/part-00000
```

gives:

```
. . .
631     1
633     1
654     1
658     1
678     1
716     1
779     1
```

The most cited patent is cited 779 times

- Command

```
$ hadoop fs –cat
 outputhis/part-00000 | head -5
```

gives:

```
1       921128
2       552246
3       380319
4       278438
5       210814
```

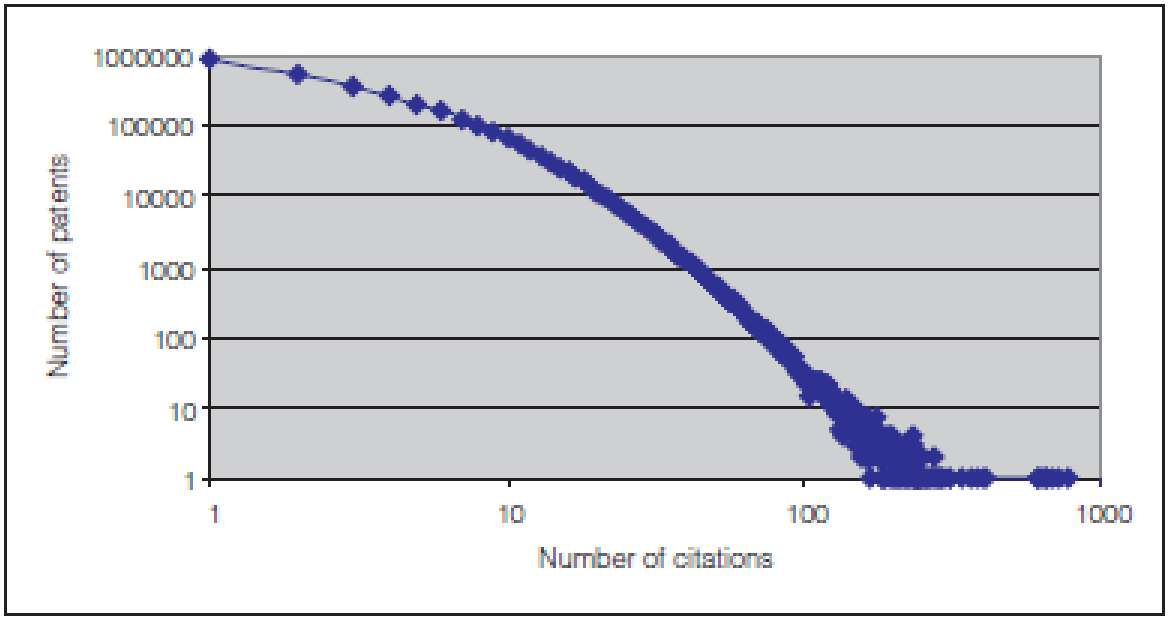
900K patents were cited only once

Histogram

- There are apparently only 258 lines in `part-00000`

```
$ hadoop fs -get outputhis/part-00000 .
$ cat part-00000 | wc -l
258
```

- While it might be less than convenient to generate a graph of these data using Hadoop, Excel could do it for us.
- Data are somewhat easier to understand if presented on log-log graph



Chaining MapReduce jobs

- We can execute the two MapReduce jobs manually one after the other, it would be more convenient to automate the execution sequence.
- We could chain two or more MapReduce jobs to run sequentially, with the output of one MapReduce job being the input to the next.
- Chaining MapReduce jobs is analogous to Unix pipes .

```
mapreduce-1 | mapreduce-2 | mapreduce-3 | ...
```

- Chaining is actually quite trivial. We could rename Mapper inner classes in `InverterCounter` and `CitationHistogram` classes to `MapClass1` and `MapClass2` respectively.
- Similarly we could rename to Reduce classes to `Reduce1` and `Reduce2`.
- All four inner classes could now reside inside a single class that we could call `ChainedHistogram`.
- Inside new class we will configure two jobs: `job1` and `job2`.
- `job1` will receive its input from HDFS directory `input`, and write its output to new HDFS directory called `temp`. `job2` will take its input from the `temp` directory and write its output to HDFS directory `output`.
- We will need two `createJob` methods: `createJob1` and `createJob2`. The content of those methods is identical to the content in classes `InverterCounter` and `CitationHistogram`, respectively.
- Once both jobs are done, we will delete the `temp` directory.

ChainedHistogram,createJob1,cleanup, main

```
private JobConf createJob1(Configuration conf, Path in, Path out) {
  JobConf job = new JobConf(conf, ChainedHistogram.class);
  job.setJobName("job1");
  FileInputFormat.setInputPaths(job, in);
  FileOutputFormat.setOutputPath(job, out);
  job.setMapperClass(MapClass1.class); job.setReducerClass(Reduce1.class);
  job.setInputFormat(KeyValueTextInputFormat.class);
  job.setOutputFormat(TextOutputFormat.class);
  job.set("key.value.separator.in.input.line", ",");
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);  return job;
}
private void cleanup(Path temp, Configuration conf)
   throws IOException {
        FileSystem fs = temp.getFileSystem(conf);
        fs.delete(temp, true);
}
public static void main(String[] args) throws Exception {
 int res = ToolRunner.run(new Configuration(), new ChainedHistogram(), args);
  System.exit(res);
}
```

Class ChainedHistogram, createJob2, run

```
private JobConf createJob2(Configuration conf, Path in, Path out) {
  JobConf job = new JobConf(conf, ChainedHistogram.class);
  job.setJobName("job2");
  FileInputFormat.setInputPaths(job, in);
  FileOutputFormat.setOutputPath(job, out);
  job.setMapperClass(MapClass2.class); job.setReducerClass(Reduce2.class);
  job.setInputFormat(KeyValueTextInputFormat.class);
  job.setOutputFormat(TextOutputFormat.class);
  job.setOutputKeyClass(IntWritable.class);
  job.setOutputValueClass(IntWritable.class); return job;
}
public int run(String[] args) throws Exception {
  Configuration conf = getConf();
  Path in = new Path(args[0]);
  Path out = new Path(args[1]);
  Path temp = new Path("chain-temp");
  JobConf job1 = createJob1(conf, in, temp);
  JobClient.runJob(job1);
  JobConf job2 = createJob2(conf, temp, out);
  JobClient.runJob(job2);
  cleanup(temp, conf);    return 0;
}
```

Migrating Code to New API

- One of the main design goals driving toward Hadoop's major 1.0 release was a stable and extensible MapReduce API.
- At the time version 0.20 was planned to be the latest release with old API and is considered a bridge between the older API (that we used in last three example) and the upcoming stable API.
- The 0.20 release supports the future API while maintaining backward-compatibility with the old API while marking it as deprecated.
- Future releases after 0.20 are supposed to stop supporting the older API.
- Almost all the changes affect only the basic MapReduce template. We will rewrite our last class `CitationHistogram` under the new API to demonstrate the changes you need to implement.

Differences between API-s

- The most noticeable change in the new API is that many classes in `org.apache.hadoop.mapred` package have been moved elsewhere.
- Many of `org.apache.hadoop.mapred` classes are now in `org.apache.hadoop.mapreduce` package.
- Some classes are now under one of the packages in `org.apache.hadoop.mapreduce.lib`.
- After you move your code to the new API, you should not have any import statements (or full references) to any classes under `org.apache.hadoop.mapred`. All of `mapred` classes are to be deprecated.

Introduction of `Context` objects

- Another noticeable change in the new API is the introduction of *`Context`* objects.
- The most immediate impact is the replacement of `OutputCollector` and `Reporter` objects used in the `map()` and `reduce()` methods.
- In new API we output key/value pairs by calling `Context.write()` instead of `OutputCollector.collect()`.
- The long-term consequences are to unify communication between your code and the MapReduce framework, and to stabilize the Mapper and Reducer API such that the basic method signatures will not change when new functionalities are added. New functionalities will only be additional methods on the context objects. Programs written before the introduction of those functionalities will be unaware of the new methods, and they will continue to compile and run against the newer releases.

org.apache.hadoop.mapreduce.Mapper.Context

- Both

org.apache.hadoop.mapreduce.Mapper and
org.apache.hadoop.mapreduce.Reducer

classes have an object of type Context, i.e.

org.apache.hadoop.mapreduce.Mapper.Context and
org.apache.hadoop.mapreduce.Reducer.Context

For some reason those classes are poorly documented .

- http://hadoop.apache.org/docs/stable/api/org/apache/hadoop/mapreduce/class-use/Mapper.Context.html
- http://hadoop.apache.org/docs/stable/api/org/apache/hadoop/mapreduce/class-use/Reducer.Context.html

New abstract classes `Mapper` and `Reducer`

- The new `map()` and `reduce()` methods are contained in new *abstract classes* `Mapper` and `Reducer`, respectively.
- New classes replace the `Mapper` and `Reducer` *interfaces* (`org.apache.hadoop.mapred.Mapper` and`org.apache.hadoop.mapred.Reducer`) in the original API.
- The new abstract classes also replace the `MapReduceBase` class, which has been deprecated.
- The new `map()` and `reduce()` methods have a couple more changes.
  - They can throw `InterruptedException` instead of only `IOException`.
  - In addition, the `reduce()` method no longer accepts the list of values as an `Iterator` but as an `Iterable`, which is easier to iterate through using Java's `foreach` syntax.

Signatures of old and new `map()` and `reduce()`

```
public static class MapClass                    //OLD
    extends MapReduceBase implements Mapper<K1, V1, K2, V2> {
    public void map(K1 key, V1 value,
                    OutputCollector<K2, V2> output, Reporter reporter)
    throws IOException { } }
public static class Reduce                      // OLD
    extends MapReduceBase implements Reducer<K2, V2, K3, V3> {
    public void reduce(K2 key, Iterator<V2> values,
        OutputCollector<K3, V3> output, Reporter reporter)
        throws IOException { } }
```

- The new API simplifies `map()` and `reduce()` somewhat:

```
public static class MapClass extends Mapper<K1, V1, K2, V2> {
     public void map(K1 key, V1 value, Context context)
     throws IOException, InterruptedException { }
}
public static class Reduce extends Reducer<K2, V2, K3, V3> {
  public void reduce(K2 key, Iterable<V2> values, Context context)
        throws IOException, InterruptedException { }
}
```

Driver Changes

- We also need to change the driver to support the new API.
- `JobConf` and `JobClient` classes have been replaced. Their functionalities have been pushed to the `Configuration` class (which was originally the parent class of `JobConf`) and a new class `Job`.
- The `Configuration` class purely configures a job, whereas the Job class defines and controls the execution of a job.
- Methods such as `setOutputKeyClass()` and `setOutputValueClass()` have moved from `JobConf` to `Job`. A job's construction and submission for execution are now under `Job`.
- Originally you would construct a job using `JobConf`:

```
JobConf job = new JobConf(conf, MyJob.class); job.setJobName("MyJob");
```

- Now we do it through `Job` object:

```
Job job = new Job(conf, "MyJob"); job.setJarByClass(MyJob.class);
```

- Previously `JobClient` submitted a job for execution:

```
JobClient.runJob(job);
```

- Now it's also done through `Job` :

```
System.exit(job.waitForCompletion(true)?0:1);
```

CitationHistogramNewApi.java, **new** vs. **old**

```
package edu.hu.bgd;  import java.io.IOException; //import java.util.Iterator;
import java.lang.InterruptedException;
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.Text;
//import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
//import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
//import org.apache.hadoop.mapred.JobClient; //import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapreduce.Job;
//import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.Mapper; //import org.apache.hadoop.mapred.Mapper;
//import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapreduce.Reducer; // import org.apache.hadoop.mapred.Reducer;
//import org.apache.hadoop.mapred.Reporter; //import org.apache.hadoop.mapred.MapReduceBase
//import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

CitationHistogramNewApi.java, **new** vs. **old**

```
public class CitationHistogramNewApi extends Configured
implements Tool {
    public static class MapClass extends
        Mapper<Text, Text, IntWritable, IntWritable> {
      private final static IntWritable uno = new IntWritable(1);
      private IntWritable citationCount = new IntWritable();
      public void map(Text key, Text value, Context context)
        throws IOException, InterruptedException {
          citationCount.set(Integer.parseInt(value.toString()));
          context.write(citationCount, uno);
        }
    }
```

CitationHistogramNewApi.java, **new** vs. **old**

```
public static class Reduce extends Reducer
        <IntWritable,IntWritable,IntWritable,IntWritable>  {
//public void reduce(IntWritable key,Iterator<IntWritable>values,
//                   OutputCollector collector, Reporter reporter)
         public void reduce(IntWritable key, Iterable<IntWritable>
                  values, Context context)
              throws IOException, InterruptedException {
           int count = 0;
           // while (values.hasNext()) {
           //     count += values.next().get();
           for (IntWritable val:values){   // Iterable allows
                count += val.get();        // for looping
           }
           context.write(key, new IntWritable(count));
       }
   }
```

CitationHistogramNewApi.java, **new** vs. **old**

```
public int run(String[] args) throws Exception {
        Configuration conf = getConf();
        //JobConf job = new JobConf(conf, CitationHistogram);
        //job.setJobName("CitationHistogram");
        Job job = new Job(conf, "CitationHistogram");
        job.setJarByClass(CitationHistogramNewApi.class);
        Path in = new Path(args[0]);
        Path out = new Path(args[1]);
        FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);
        job.setJobName("CitationHistogramNewApi");
        job.setMapperClass(MapClass.class);
        job.setReducerClass(Reduce.class);
        //job.setInputFormat(KeyValueTextInputFormat.class);
        job.setInputFormatClass(KeyValueTextInputFormat.class);
        //job.setOutputFormat(TextOutputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);
        //JobClient.runJob(job);
        System.exit(job.waitForCompletion(true)?0:1);
        return 0;
    }
```

CitationHistogramNewApi.java, **new** vs. **old**

```
public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(),
                        new CitationHistogramNewApi(), args);
        System.exit(res);
    }
}
```

- New class `CitationHistogramNewApi` performs identically as `CitationHistogram` class.