Programming Assigment 1

Big Data System

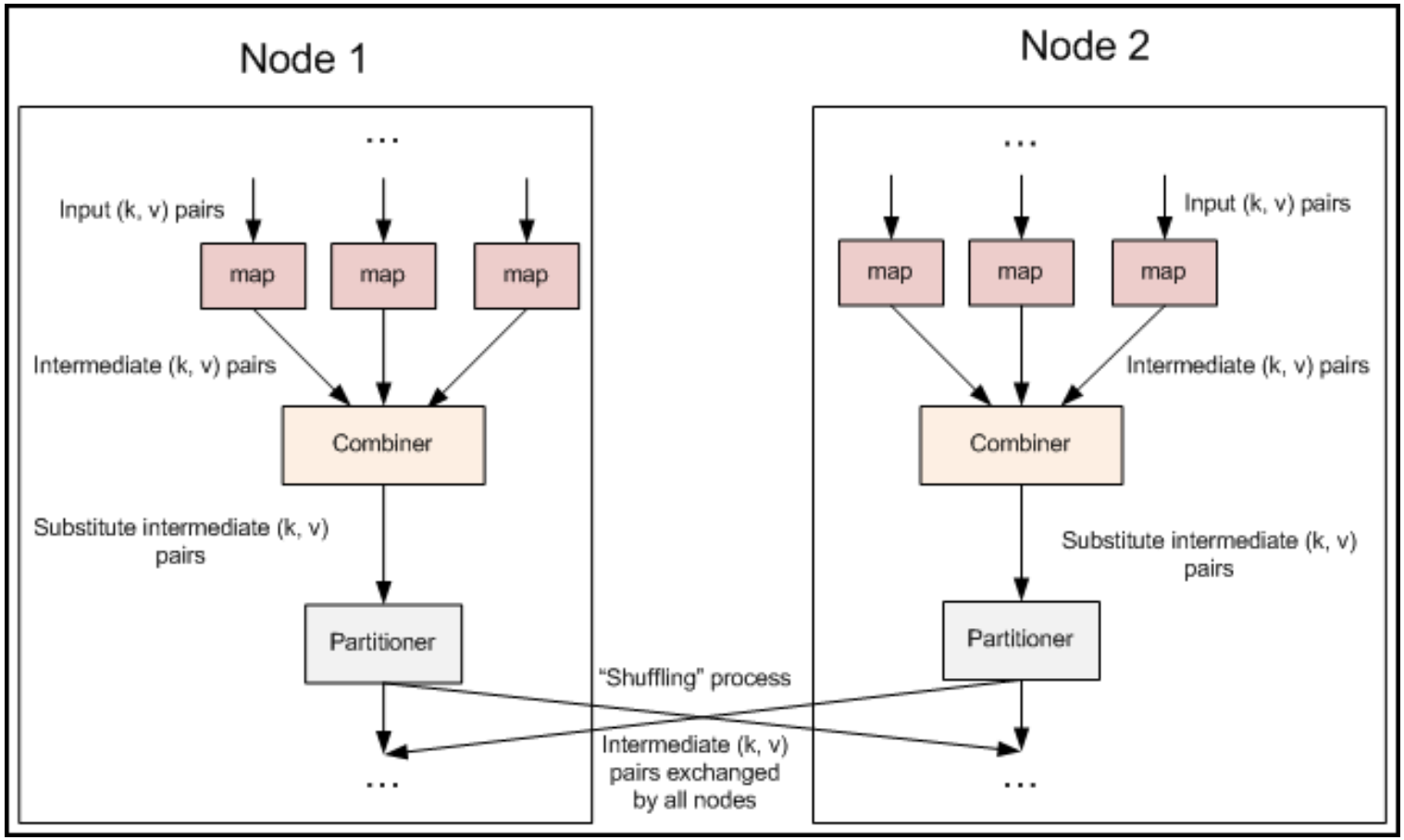
Description

- Implement the following two algorithms in map-reduce
  - Sort
  - PageRank

Sort

- Based on WordCount.java
  - Use the shuffle&sort from Hadoop System
  - The output from reducers will be sorted by key autometically by Hadoop System

Sort


Node 1
Node 2
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Combiner
Substitute intermediate (k, v) pairs
Partitioner
"Shuffling" process
Intermediate (k, v) pairs exchanged by all nodes
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Combiner
Substitute intermediate (k, v) pairs
Partitioner

MapReduce Example Walkthrough: WordCount

```
public static class MapClass extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {
```

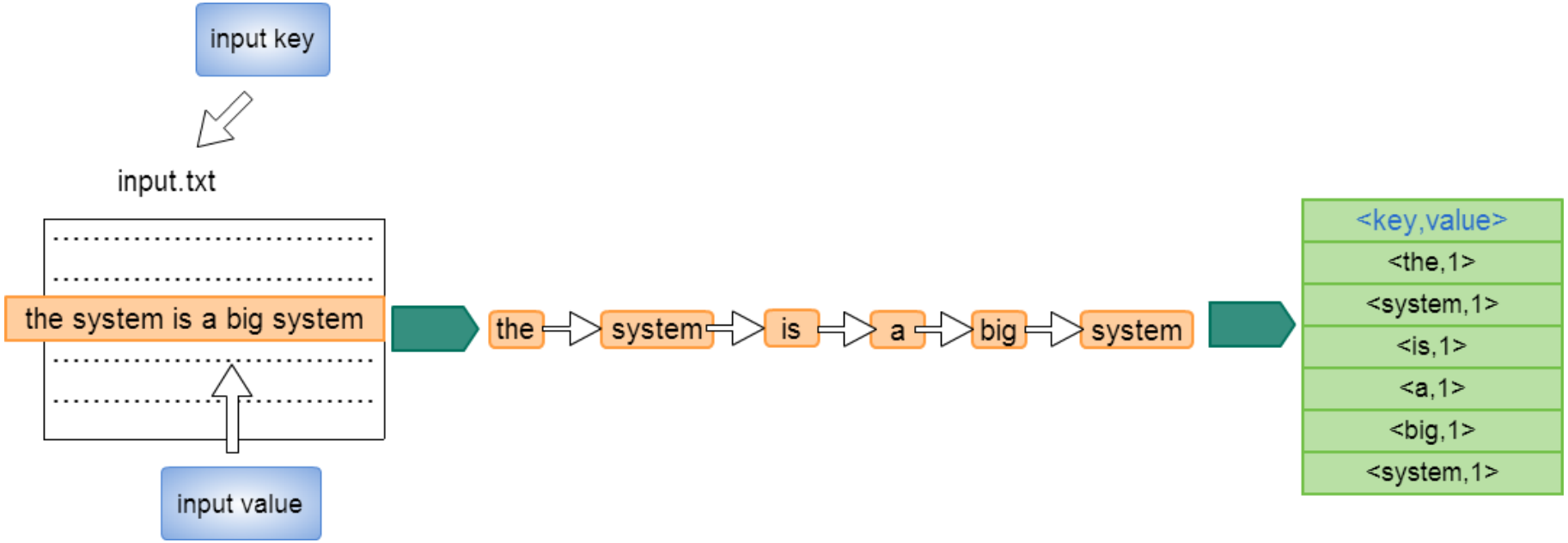
Input type: Key, Value **Output type: Key, Value**

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

  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output,
                  Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}
```

Collect result for mapper stage

MapReduce Example Walkthrough: WordCount



MapReduce Example Walkthrough: WordCount

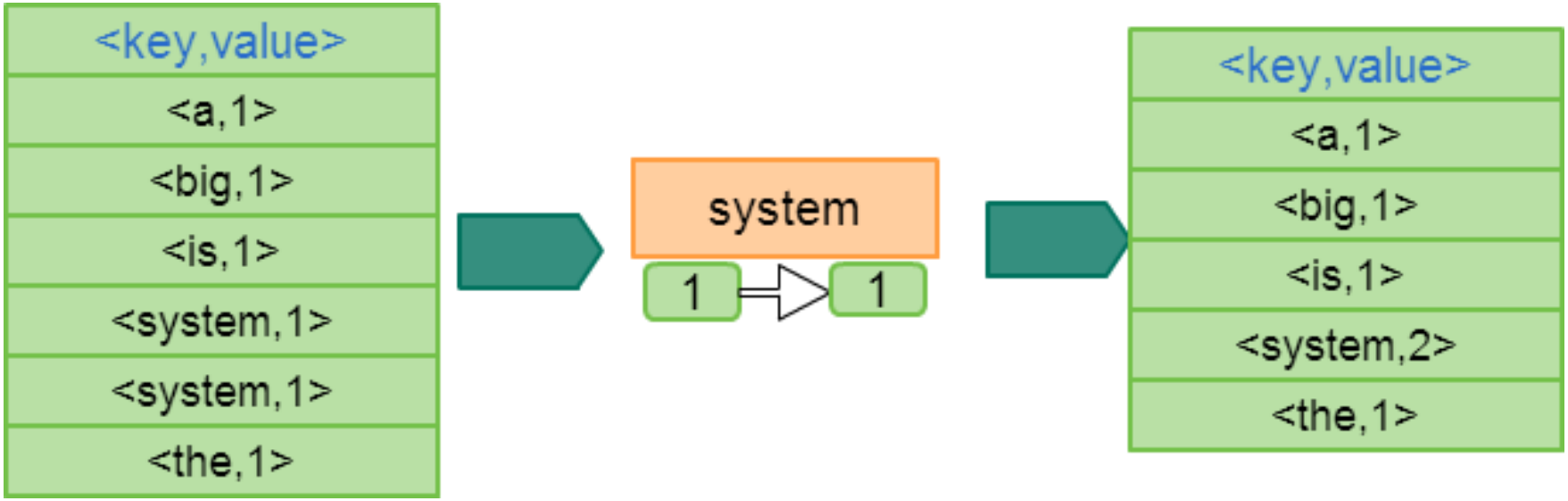
```
public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {
```

Input type: Key, Value **Output type: Key, Value**

```
    public void reduce(Text key, Iterator<IntWritable> values,
                       OutputCollector<Text, IntWritable> output,
                       Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) {
        sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
    }
  }
```

Output type in Map stage must be same as inut type in Reduce stage!

MapReduce Example Walkthrough: WordCount



MapReduce Example Walkthrough: WordCount

```
public int run(String[] args) throws Exception {
  JobConf conf = new JobConf(getConf(), WordCount.class);
  conf.setJobName("wordcount");

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

  conf.setMapperClass(MapClass.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);
  …
  conf.setInputFormat(FileInputFormat.class);
  conf.setOutputFormat(FileOutputFormat.class);
  FileInputFormat.setInputPaths(conf, other_args.get(0));
  FileOutputFormat.setOutputPath(conf, new Path(other_args.get(1)));

  JobClient.runJob(conf);
  return 0;
}
```

We don’t need combiner when writing PageRank and Sort.

Type of Input & Output

- If your mapper output (key, value) type is different from reducer (key, value) type, you may need to set it using
  - JobConf::setMapOutputKeyClass
  - JobConf::setMapOutputValueClass
- FileInputFormat
  - Read all files from input paths, and split them by lines
  - Feed lines into mapper
- FileOutputFormat
  - Output key, value separated with <tab>
- You can override FileInputFormat::createRecordReader, FileOutputFormat::getRecordWriter to customize input/output format, but we sugguest you just use the default one.

Result from WordCount

```
road,    1
robbery 1
said     3
say      1
scene    2
sending 1
senior   1
seven    1
shooting         1
shootout         1
shoppers         1
shoppers.        1
shot.    1
someone 1
still    1
store,   1
story    2
strike   1
surrounded       1
taken    1
terrorist        1
that     1
```

PageRank

- The importance(PageRank) of a page is the sum of PageRank of all the pages that points to it.

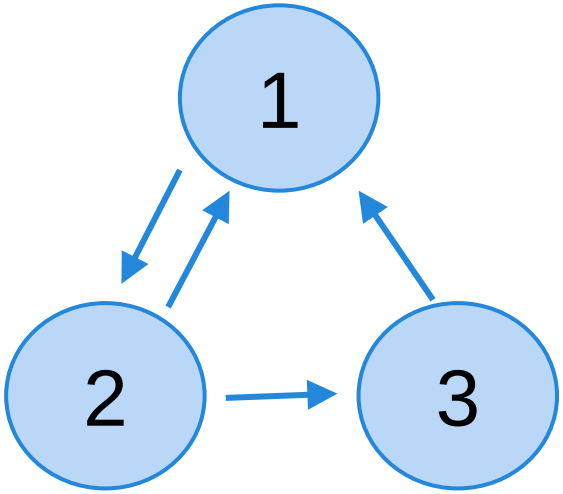
$$PR(p_i; t+1) = 1-d+d \sum_{p_j \in M(p_i)} \frac{PR(p_j; t)}{O(p_j)}$$

- d is a damping factor: Google suggest 0.85
- Need to run several iterations until it’s stable. For this homework we will pass the number of iteration as parameter.

PageRank Example

- Input

1	0.3	2
2	0.3	1 3
3	0.3	1



$$PR(p_i; t+1) = 1-d+d \sum_{p_j \in M(p_i)} \frac{PR(p_j; t)}{O(p_j)}$$

- Single Iteration:
  - P(1) = (1 - d)+ d (0.3 / 2 + 0.3 / 1)

    from vertex 2 from vertex 3
  - P(2) = (1 - d)+ d (0.3 / 1)
  - P(3) = (1 - d)+ d (0.3 / 2)

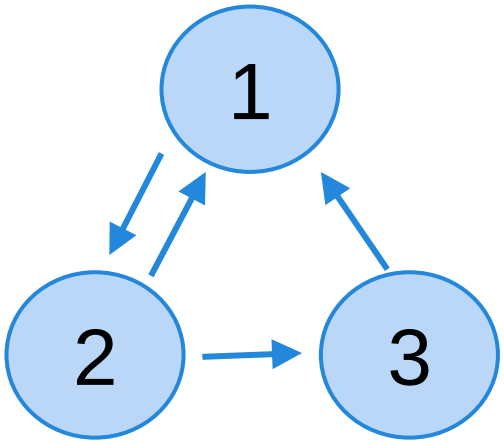
PageRank in MapReduce

- Mapper:
  - Input: (node, (rank, outlinks))
    key value
  - Output:
    - (node, outlinks)
    - (outlink1, contrib1), (outlink2, contrib2) …
- Reducer:
  - Input:
    - (node, outlinks)
    - (node, contrib1), (node, contrib2) ...
  - Output: (node, (rank, outlinks))
- Note that since we need to run multiple iterations, the input format of mapper and ouput format of reducer must be the same.

PageRank Example: Mapper

- Input

1	0.3	2
2	0.3	1 3
3	0.3	1



- Mapper: (output)(key,value)

(2, 0.3 / 1) (1, 2)

(1, 0.3 / 2) (3, 0.3 / 2) (2, 1 3)

(1, 0.3 / 1) (3, 1)

PageRank Example: Reducer

- Input

  (1, 2) (1, 0.3 / 2) (1, 0.3 / 1)

  (2, 1 3) (2, 0.3 / 1)

  (3, 1) (3, 0.3 / 2)

- Ruducer (Output)

key	value	
1	(1 - d) + d (0.3 / 2 + 0.3 / 1))	2
2	(1 - d) + d (0.3 / 1))	1 3
3	(1 - d) + d (0.3 / 2))	1

Input Data Format

- example.graph

```
1<tab>1.0<tab>2 3
2<tab>1.0<tab>1 3 4
3<tab>1.0<tab>5
4<tab>1.0<tab>1 3
5<tab>1.0<tab>2 4
```

- Each line represents (node, pagerank, outlinks) saparated by tab.

- Each outlink is separated by space

Hints

- How to run iteration?
  - Remove input directory
    - use org.apache.hadoop.fs.FileSystem::delete
  - Assign output directory to be new input directory
    - FileInputFormat::setInputPaths
    - FileOutputFormat::setOutputPath

Compile & Run

- compile
  - javac -classpath hadoop-*-core.jar -d MyJava MyCode.java
- pack
  - jar -cvf MyJar.jar -C MyJava .
- run
  - bin/hadoop jar MyJar.jar MyCode HDFS_Input/ HDFS_Output/

http://hadoop.apache.org/docs/stable/mapred_tutorial.html#Example%3A+WordCount+v1.0

Compile & Run

- ./PageRankCompile.sh
  - We provide a environment variable $CLASSPATH that includes hadoop-core.jar

- ./PageRankRun.sh <*iter*> <*input*> <*output*>
  - **iter**: number of PageRank iteration
  - **input**: input directory, contains single graph file
  - **output**: output directory name
  - Note: you can assume the hadoop command exists in the PATH.

Compile & Run

- ./SortCompile.sh
  - We provide a environment variable $CLASSPATH that includes hadoop-core.jar

- ./SortRun.sh <*input*> <*output*>
  - **input**: input directory, contains single list file
  - **output**: output directory name
  - Note: you can assume the hadoop command exists in the PATH.

Test Data

- small.graph
  - small.graph.18.out is the result of 18 iterations.
- big.graph
  - big.graph.10.out is the result of 10 iterations.
- small.list
  - test data for sorting, small dataset
- big.list
  - test data for sorting, big dataset

Requirements

- Sort (40%)
  - Sort.java
  - SortCompile.sh & SortRun.sh (10% penalty)
- PageRank (50%)
  - PageRank.java
  - PageRankCompile.sh & PageRankRun.sh (10% penalty)
- Report (10%)
  - Implementation Description
  - Problems Encountered
  - At most 1 A4 page in PDF format

Submission

Upload a zip file named in “Your School ID"

Use “r00922000” as an example :

```
r00922000.zip
└── r00922000
    ├── Report.pdf
    ├── PageRankCompile.sh
    ├── PageRankRun.sh
    ├── SortCompile.sh
    ├── SortRun.sh
    └── Other source files
```

Submission

- Due: 2013/ 10/ 17 18:30:00
- Late submissions lose 10% per day
- Plagiarism will not be tolerated

Questions

- If you have any questions, please ask it at CEIBA
- Do not send email directly to the professor or TAs unless you have other questions not relating to the homework.