“Map Reduce” Computing Paradigm .2

pm jat @ daiict

Map Reduce - Recap

- Map Reduce is a "**Programming Abstraction**" over "Computers in a Cluster"
- In the approach Computation is defined in terms of Map and Reduce tasks.
- Application programmers only define Map and Reduce functions, "rest of work" is taken care by Map-Reduce infrastructure like Hadoop.
  - What is "Rest of Work"? Delegation of map-reduce tasks, communication, data distribution, replication(fault tolerance), input/output management, data shuffling, MR Job Tracking, coordination, etc.

Map Reduce - Recap

- All Input and Outputs to Map and Reduce functions are always “Key-Value” pair.
- Identifying Key and Value for map-reduce tasks is central to the map-reduce solution for a computation.
- We shall discuss them in various examples that we see here.
- Map and Reduce tasks are performed in parallel on several computers in the cluster. Map and Reduce tasks are however sequenced in that sense that reduce processes output of map function

Map Reduce - Recap

- A number of computers perform map task, called mappers. Another set of computers perform reduce task are called reducers.
- Output of Map tasks are "**shuffled**" to the reducers.
- Target reducer for a map output is determined by applying a "partitioning hash function" on key of map function output.
- Reducer receives all values of a key.
- Input to reduce function is <key, and value-list>
- Reducer typically aggregates input and generates a key-value pair as output that is finally made available on DFS.

Map Reduce - Recap

- Components of a Map-Reduce infrastructure (like Hadoop)
  - Distributed File System: Hadoop Distributed File System (HDFS) in case of Hadoop.
    - DFS keeps data file distributed and replicated on multiple computing nodes (computers in a cluster are called computing nodes)
  - Map-Reduce Library
  - Map Reducer Job Tracker (at Master Node)

Map Reduce Programming Setup

- To get a full picture, we may require
  - Downloading and setting up Hadoop
  - Creating a computing cluster (single node or multi-node)
  - Create map-reduce functions, creating a jar file, submit that to job tracker at master node, etc.
- In our labs, we are bypassing the setup. Eclipse acts as simulator for us.
- It is OK, as far as scope of this course is concerned!

Example #1: Max Temperature

- **Input**: a data file without delimiter). Has many more data than year and temperature

```
0029029070999991901010106004+64333+023450FM-12+000599999V0202701N015919999999N0000001N9-00781
0029029070999991901010113004+64333+023450FM-12+000599999V0202901N008219999999N0000001N9-00721
0029029070999991901010120004+64333+023450FM-12+000599999V0209991C000019999999N0000001N9-00941
0029029070999991901010206004+64333+023450FM-12+000599999V0201801N008219999999N0000001N9-00611
0029029070999991901010213004+64333+023450FM-12+000599999V0201801N009819999999N0000001N9-00561
0029029070999991901010220004+64333+023450FM-12+000599999V0201801N009819999999N0000001N9-00281
```

- **Output**: year wise maximum temperature.
- In SQL terms, we compute following on a plain text data file (not a DB table)

  "`select year, max(temperature)`
  `from weather group by year`"

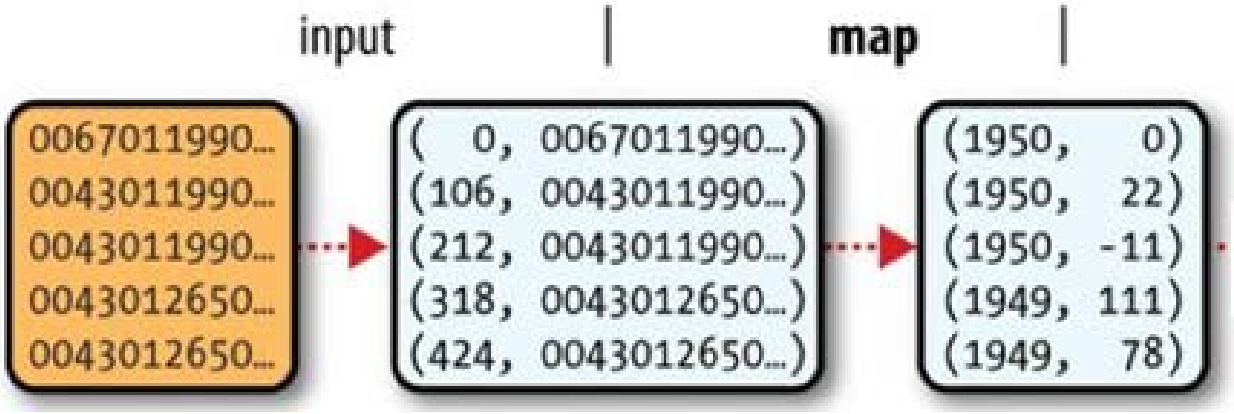
Map Task

input: <key, value>
<recno, record>

output: <key, value>
<year, temperature>

Max Temp Map function

```
void map(recno, line){
    String year = line.substring(15, 19);
    int temp = line.substring(88, 92);
    String quality = line.substring(92, 93);
    if (quality.matches("[01459]"))
        output(year, temp);
}
```



Reduce Task

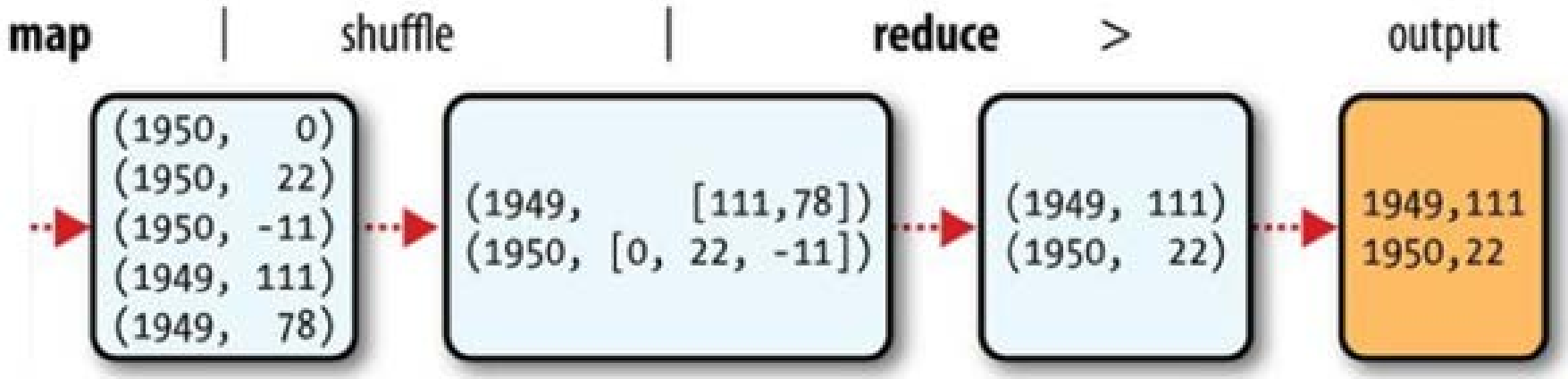
input: <key, value-list>
<year, list of temperatures>

output: <key, value>
<year, max(temperatures)>

For every distinct "key" reducer function will be called

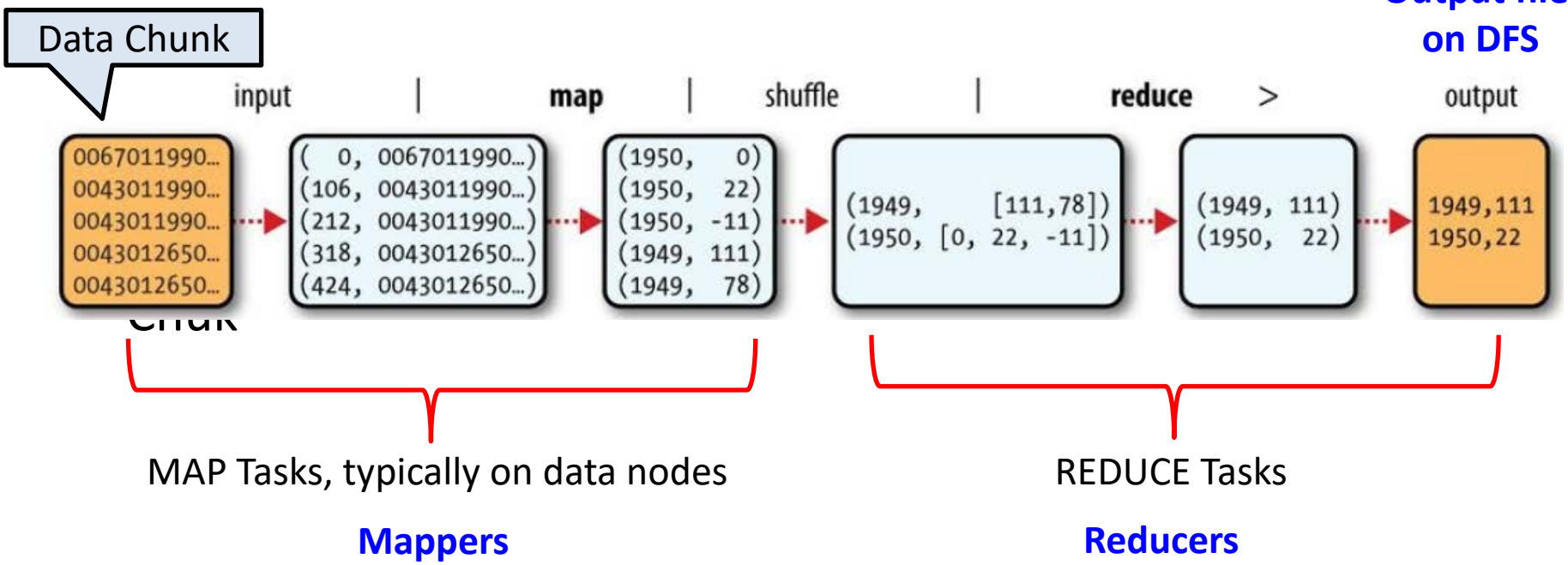
Max Temp Reduce function

```
void reduce(year, values) {
    int maxValue = MIN_VALUE;
    for (value : values)
        maxValue = max(maxValue, value);
    output(year, maxValue);
}
```

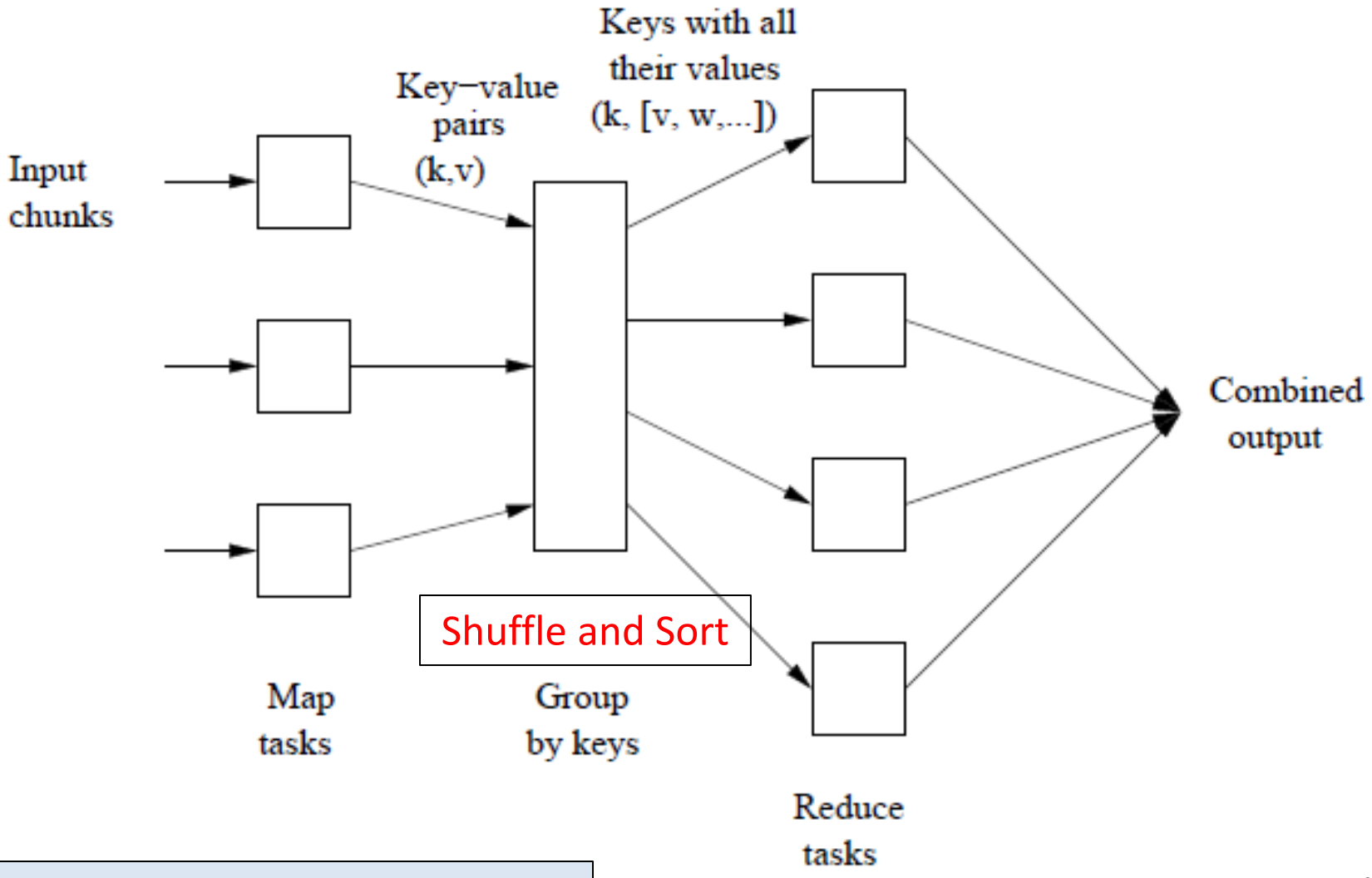


Compute maximum temperature[5]

- Computation is performed by Map and Reduce tasks and in a pipeline. Here is how it goes-



Schematic of a Map Reduce computation



http://infolab.stanford.edu/~ullman/mmds/ch2.pdf

MR Example #2: Word Count

- Word count from a huge document corpus
  - A huge set of documents stored on HDFS
- Want to find out word count in the corpus
  - i.e. aggregated over all documents

MR Example #2: Word Count

- In pseudo code!

```
//key:lineno, value: line
void map(line#, line){
    words = tokenize(line);
    for word token in words
        emit(word, 1);
    }
}
```

MAP function

input: <key, value>
<line#, line>

output: <key, value>
<word, 1>

#note this may be repeated for all files in the specified directory; in such a case input is directory name rather than a file name

MR Example #2: Word Count

- In pseudo code!

```
//key:word, counts
void reduce(word, counts) {
    int sum = 0;
    for each value in counts
        sum += value;
    emit(word, sum);
}
```

Reduce function
input: <key, value-list>
<word, <1,1,1,1,1,1>>
output: <key, value>
<word, sum(1's)>

MR Example #2: Word Count

- In pseudo code!

```
//key:lineno, value: line
void map(line#, line){
    words = tokenize(line);
    for word token in words
        emit(word, 1);
    }
}
```

MAP function

input: <key, value>
<line#, line>

output: <key, value>
<word, 1>

#note this may be repeated for all files in the specified directory; in such a case input is directory name rather than a file name

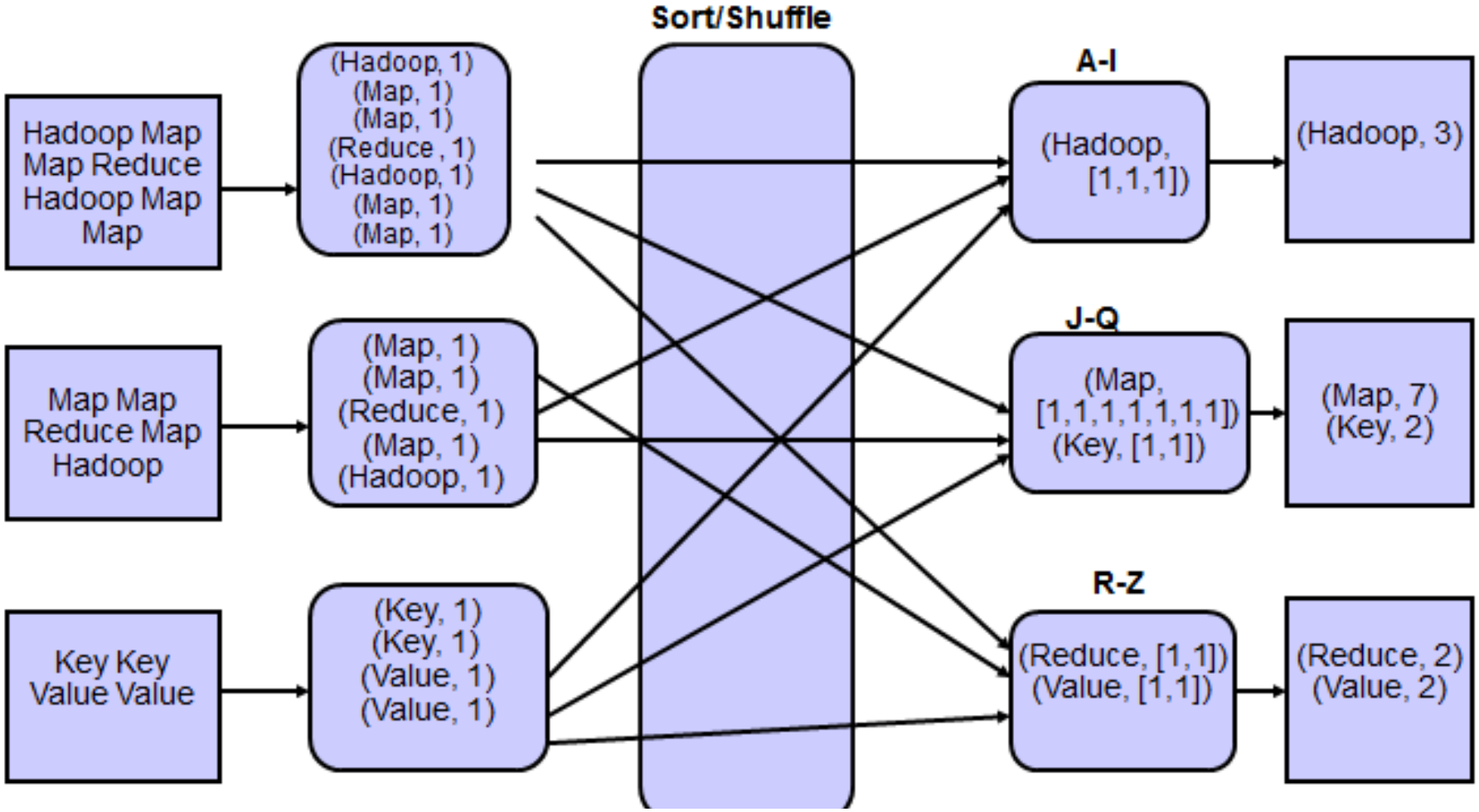
```
//key:word, counts
void reduce(word, counts) {
    int sum = 0;
    for each value in counts
        sum += value;
    emit(word, sum);
}
```

Reduce function

input: <key, value-list>
<word, <1,1,1,1,1,1>>

output: <key, value>
<word, sum(1's)>

Word-Count



3 Mappers and 3 Reducers

Relation Algebra Operations on MR

- Selection and Projection
  - Have selection condition applied in Map functions
  - It can be map only Job. There should be ways of submitting map only jobs.
  - Value here is a Tuple. Hadoop provides a data type WritableTuple for such cases.

```
void map(rid, record){

    tokens = record.split(",")
    ssn = tokens[ssn_pos]
    fname = tokens[fname_pos]
    dno = tokens[dno_pos]
    salary=row[salary_pos]
    if dno=5 or salary >= 50000
        output(ssn, <ssn, fname, salary>);
}
```

Aggregate Operations using MR

- Mapper outputs:

  Key: Grouping Attribute, Value: Aggregating Attribute Value

- Reducer outputs:

  Key: Grouping Attribute, Value: Aggregated value

MR Example #3: Max Salary

- Suppose there is a huge employee file, and we would want to compute department-wise sum of salary, that is equivalent to `select dno, sum(salary) from employee group by dno;`

```
void map(recid, record){

    dno = record[pos_dno];
    salary = record[pos_salary]
    if salary != NULL
        output(dno, salary)
}
```

input: <key, value>
<recid, record>
output: <key, value>
<dno, salary>

Grouping attribute is key here

```
void reduce(dno, salaries) {
    sum = 0;
    for (salary : salaries)
        sum += salary
    output(dno, sum);
}
```

input: <key, value-list>
<dno, salary-list>
output: <key, value>
<dno, sum(salaries)>

MR Example #3: AVG Salary

- Compute department-wise average salary, that is equivalent to select dno, avg(salary) from employee group by dno;

```
void map(rowid, row){
    dno = row.dno;
    salary = row.salary;
    if salary != NULL
        output(dno, salary);
}
```

input: <key, value>
<recid, record>
output: <key, value>
<dno, salary>

input: <key, value-list>
<dno, salary-list>
output: <key, value>
<dno, average(salaries)>

```
void reduce(dno, salaries) {
    sum = 0;
    n = 0;
    for (salary : salaries) {
        sum += salary;
        n++;
    }
    avg_sal = sum/n;
    output(dno, avg_sal);
}
```

Map-Reduce **Combiners**

- Often a Map task will produce many pairs of the form $(k,v_1), (k,v_2), \ldots$ for the same key k
- For huge files, this will require moving large amount of data to reducers over the network; and end engaging more number of reducer. Ultimately making computation slow!
- This can be saved by doing some aggregation at local level only, i.e. the node where maps function runs.
- Map Reduce framework allows specifying a "**combiner**" for this purpose!

  $\text{combine}(k, \text{list}(v_1)) \rightarrow (k,v_2)$

- This is comes as third function in Map-Reduce. However this is basically reduce only at local level

Video Lessons by Anand Rajaraman, http://www.mmds.org

Word Count with combiner

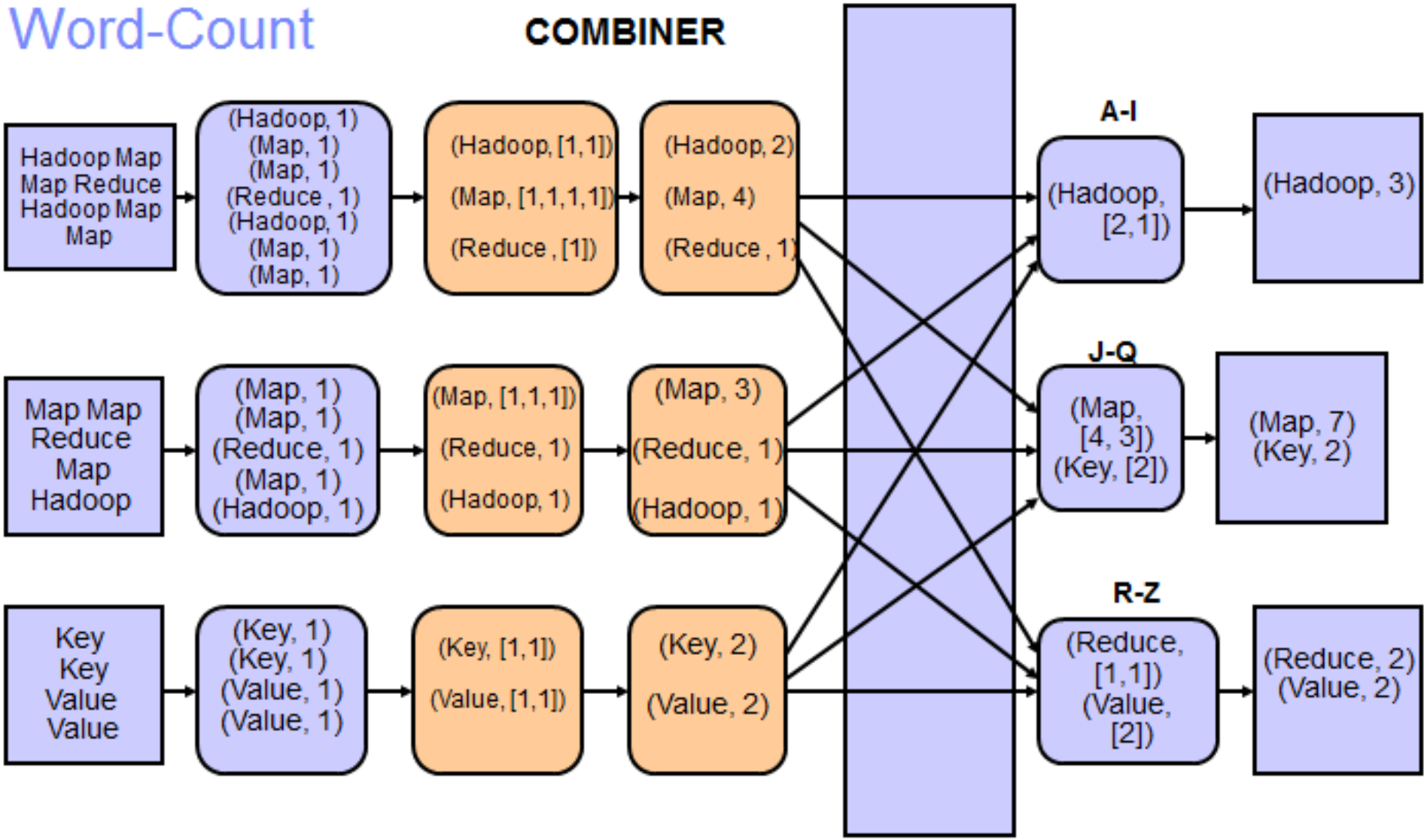
- Word count with combiner

```
//key:lineno, value: line
void map(line#, line){
    words = tokenize(line);
    for word token in words
        emit(word, 1);
    }
}
```

```
//key:word, counts
void reduce(word, counts) {
    int sum = 0;
    for each value in counts
        sum += value;
    emit(word, sum);
}
```

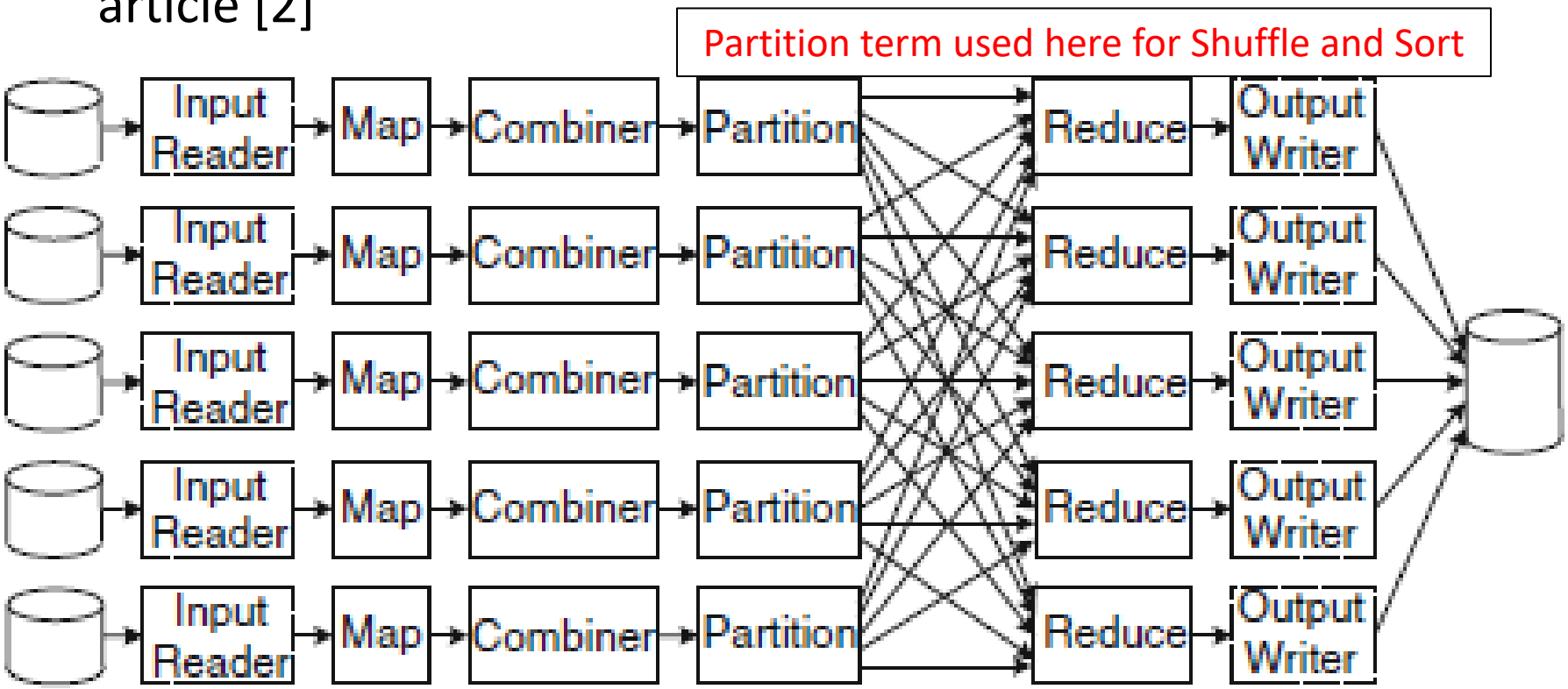
```
//key:word, counts
void combine(word, counts) {
    int sum = 0;
    for each value in counts
        sum += value;
    emit(word, sum);
}
```

Word-Count


COMBINER
Hadoop Map
Map Reduce
Hadoop Map
Map
(Hadoop, 1)
(Map, 1)
(Map, 1)
(Reduce , 1)
(Hadoop, 1)
(Map, 1)
(Map, 1)
(Hadoop, [1,1])
(Map, [1,1,1,1])
(Reduce , [1])
(Hadoop, 2)
(Map, 4)
(Reduce , 1)
A-I
(Hadoop, [2,1])
(Hadoop, 3)
Map Map
Reduce
Map
Hadoop
(Map, 1)
(Map, 1)
(Reduce, 1)
(Map, 1)
(Hadoop, 1)
(Map, [1,1,1])
(Reduce, 1)
(Hadoop, 1)
(Map, 3)
(Reduce, 1)
(Hadoop, 1)
J-Q
(Map, [4, 3])
(Key, [2])
(Map, 7)
(Key, 2)
Key
Key
Value
Value
(Key, 1)
(Key, 1)
(Value, 1)
(Value, 1)
(Key, [1,1])
(Value, [1,1])
(Key, 2)
(Value, 2)
R-Z
(Reduce, [1,1])
(Value, [2])
(Reduce, 2)
(Value, 2)

Map-Reduce dataflow[2]

- Let this be bit more detailed sketch of MR data flow from a article [2]



Combiner in SUM

- Combiner for SUM: select dno, sum(salary) from employee group by dno;

```
void map(rowid, row){
    dno = row.dno;
    salary = row.salary
    if salary != NULL
        output(dno, salary);
}
```

```
void combine(dno, salaries) {
    sum = 0;
    for (salary : salaries)
        sum += salary
    output(dno, sum);
}
```

```
void reduce(dno, salaries) {
    sum = 0;
    for (salary : salaries)
        sum += salary
    output(dno, sum);
}
```

Map-Reduce **Combiners**

- In many cases, Combiner is usually same as the reduce function.
- This however works only when reduce function is commutative and associative. SUM is, where as AVERAGE is not.
- However we can have a trick used in Combiner: we output sum and count at combiner!
- Example next:

Combiner in Average

- Combiner for AVG

  select dno, avg(salary) from employee group by dno;

```
void map(rowid, row){
    dno = row.dno;
    salary = row.salary;
    if salary != NULL
        output(dno, <salary,1>);
}
```

```
void reduce(dno, values_pairs) {
    sum = 0;
    count = 0;
    for (value : value_pairs) {
        sum += value.salary;
        count += value.count;
    }
    avg_sal = sum/count;
    output(dno, avg_sal);
}
```

```
void combine(dno, values_pair) {
    sum = 0;
    count = 0;
    for (value : value_pair) {
        sum += value.salary;
        count += value.count;
    }
    output(dno, <sum,count>);
}
```

Writing Map-Reduce programs!

- Find out how do we write and specify map and reduce functions
- You need a “driver program” for running a map reduce job, find out how it is done
- Basic data types found in a programming language can not be used for performing read/write operations on distributed fie system.
- Find out various data types that are used in Hadoop for this purpose. For example- IntWritable, DoubleWritable, Text, or so.
- Often you also require specifying, input processing format, that is how data chunks are to be read and processed!
- Sometimes, we also require specifying customized Comparators and Partitioning functions!
- Let us try decoding some Hadoop Map-Reduce programs.

Write first MR Java program on Hadoop

- Hadoop Tutorial at https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html is the nice place to begin with.
- Word Count example there is like Hello Word for hadoop!
- It has a
  - A Mapper class
  - A Reducer Class
  - A Driver “main” function

Write first MR Java program

```
//Job Class
public class WordCount {

   //Mapper Class
   public static class TokenizerMapper
         extends Mapper<Object, Text, Text, IntWritable>{

   //Reducer Class
   public static class IntSumReducer
         extends Reducer<Text,IntWritable,Text,IntWritable> {

   //Driver Function
   public static void main(String[] args) throws Exception {
}
```

MR Job "Word Count" Driver function

```
//Driver Function
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class); //User Library
  job.setMapperClass(TokenizerMapper.class); //Specify Mapper
  job.setCombinerClass(IntSumReducer.class); //Specify Combiner
  job.setReducerClass(IntSumReducer.class); //Specify Reducer
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  //Specify: Input/output locations (Can be File or Directory)
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  boolean success = job.waitForCompletion(true);
  System.out.println(success);
}
```

Hadoop MR classes

- Hadoop classes used in Word Count
- Good idea to get first hand information about these classes

```
org.apache.hadoop.conf.Configuration;
org.apache.hadoop.fs.Path;
org.apache.hadoop.io.IntWritable;
org.apache.hadoop.io.Text;
org.apache.hadoop.mapreduce.Job;
org.apache.hadoop.mapreduce.Mapper;
org.apache.hadoop.mapreduce.Reducer;
org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

“WordCount” – Mapper Class

```
public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

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

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

"Word Count" – Reducer Class

```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

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

Writing Map-Reduce programs for a problem!

- In map-reduce program, we always require breaking a problem in map-reduce tasks.
- In some cases like performing aggregate operations on a data file, map-reduce programming quiet straight forward.
- However as logic deviates from this, require iterations etc, solving it through map-reduce becomes bit challenging.
- Let us look into more examples!
  - JOIN, SORT, TOP K, ??

Sources/References

[1] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: Simplified data processing on large clusters." (2004)

[2] Doulkeridis, Christos, and Kjetil NØrvåg. "A survey of large-scale analytical query processing in MapReduce." *The VLDB Journal—The International Journal on Very Large Data Bases* 23.3 (2014): 355-380.

[3] Parsian, Mahmoud. Data algorithms: recipes for scaling up with Hadoop and Spark, O'Reilly, 2015

[4] Video lessons by Anand Rajaraman on Map-Reduce at mmds.org

[5] White, Tom. “Hadoop: The definitive guide“, 4th ed, O'Reilly Media, Inc., 2015.