Dr. Ambarish G. Mohapatra

IOT & Applications

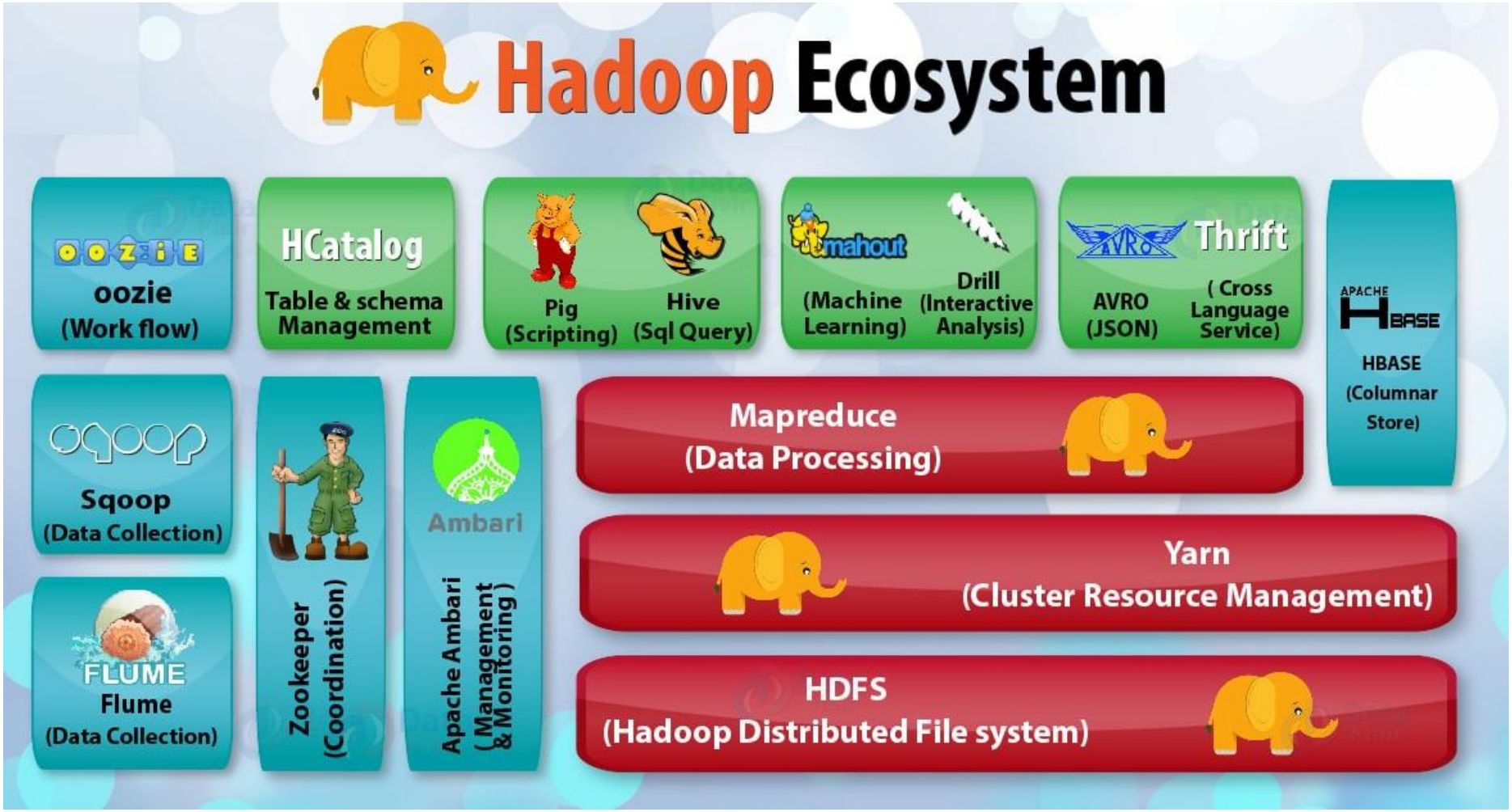
Module – 5
Class-2

Introduction to Data Analytics: MapReduce

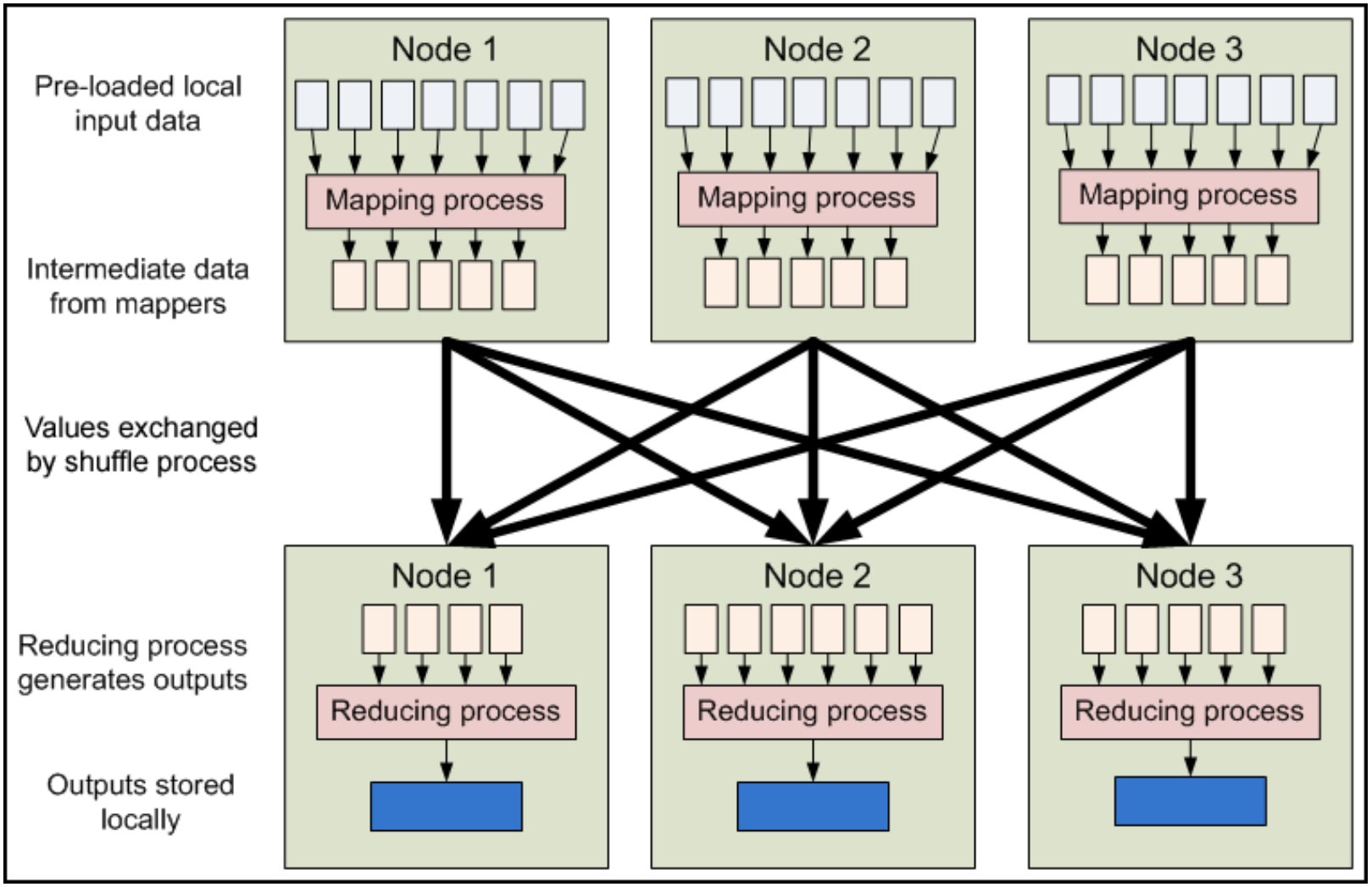
Silicon ...beyond teaching
Silicon Institute of Technology
Bhubaneswar

What is MapReduce?

- MapReduce is a programming model for efficient distributed computing
- It works like a Unix pipeline
- **MapReduce** is a software framework and programming model used for processing huge amounts of data. **MapReduce** program work in two phases, namely, Map and Reduce.
- Map tasks deal with splitting and mapping of data while Reduce tasks shuffle and reduce the data.
- A good fit for a lot of applications
  - Log processing
  - Web index building


Hadoop Ecosystem
oozie
(Work flow)
HCatalog
Table & schema Management
Pig
(Scripting)
Hive
(Sql Query)
mahout
(Machine Learning)
Drill
(Interactive Analysis)
AVRO
(JSON)
Thrift
(Cross Language Service)
APACHE HBASE
HBASE
(Columnar Store)
Sqoop
(Data Collection)
Zookeeper
(Coordination)
Ambari
Apache Ambari
(Management & Monitoring)
Mapreduce
(Data Processing)
Yarn
(Cluster Resource Management)
FLUME
Flume
(Data Collection)
HDFS
(Hadoop Distributed File system)

MapReduce - Dataflow



Different phases of MapReduce in BigData

The data goes through the following phases of MapReduce in Big Data

Input Splits:

- An input to a MapReduce in Big Data job is divided into fixed-size pieces called input splits. Input split is a chunk of the input that is consumed by a single map.

Mapping:

- This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. For example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

Shuffling:

- This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubbed together along with their respective frequency.

Reducing:

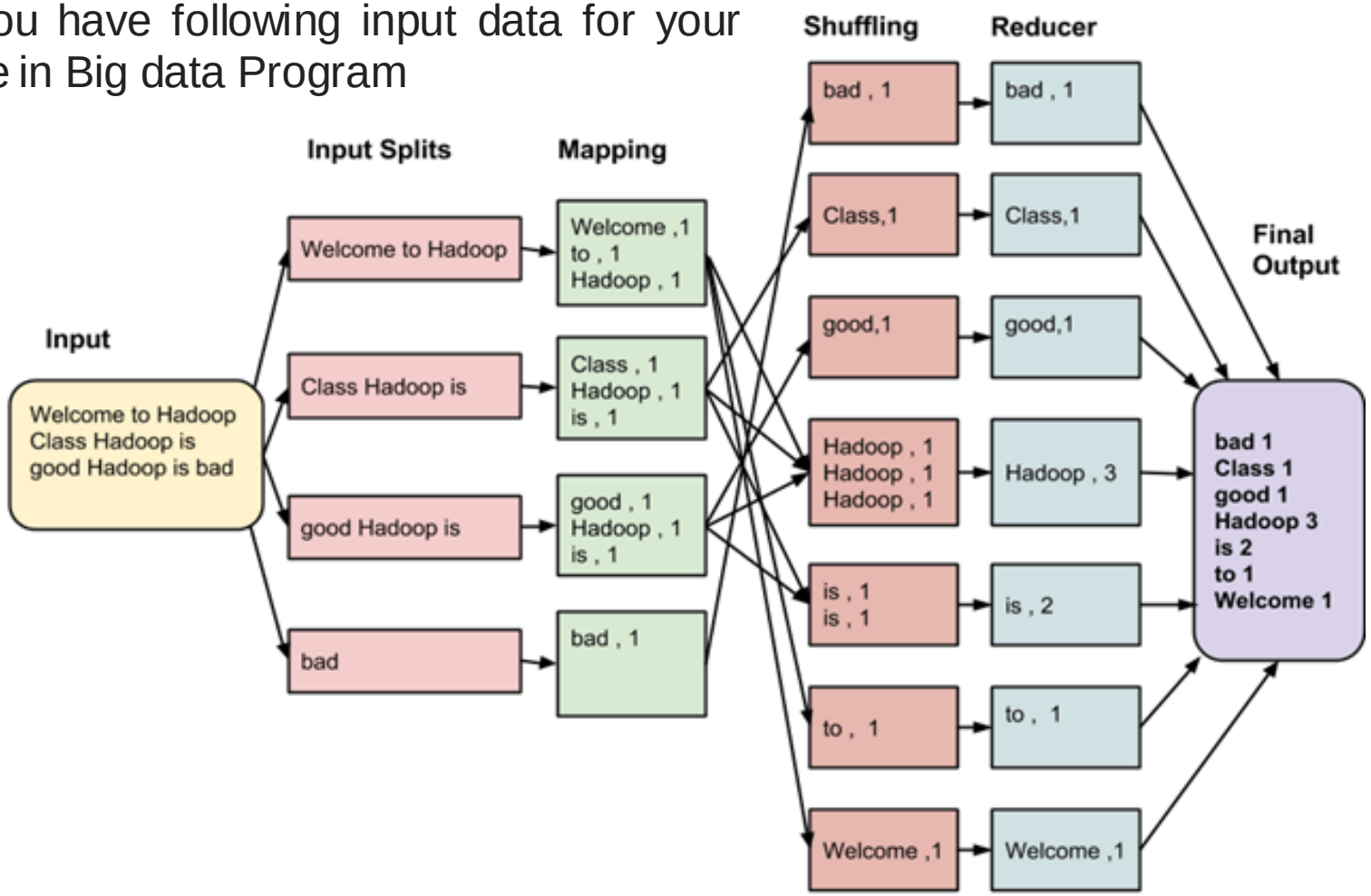
- In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

MapReduce - Features

- Fine grained Map and Reduce tasks
  - Improved load balancing
  - Faster recovery from failed tasks
- Automatic re-execution on failure
  - In a large cluster, some nodes are always slow or flaky
  - Framework re-executes failed tasks
- Locality optimizations
  - With large data, bandwidth to data is a problem
  - Map-Reduce + HDFS is a very effective solution
  - Map-Reduce queries HDFS for locations of input data
  - Map tasks are scheduled close to the inputs when possible

MapReduce Architecture: Word count example

Let's understand with a MapReduce example–
Consider you have following input data for your MapReduce in Big data Program



Output of MapReduce: Word count example

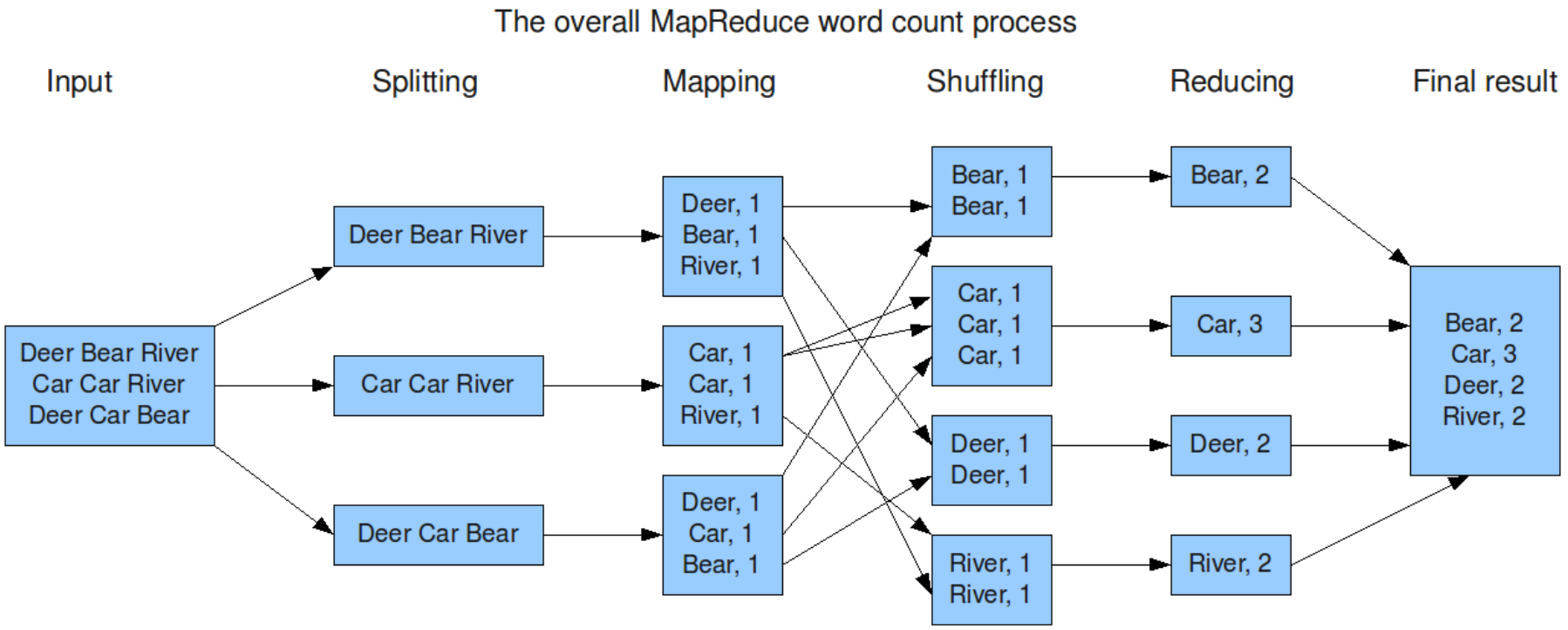
The final output of the MapReduce task is

bad	1
Class	1
good	1
Hadoop	3
is	2
to	1
Welcome	1

Word Count Example

- Mapper
  - Input: value: lines of text of input
  - Output: key: word, value: 1
- Reducer
  - Input: key: word, value: set of counts
  - Output: key: word, value: sum
- Launching program
  - Defines this job
  - Submits job to cluster

Word Count Dataflow



Word Count Mapper

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

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

Word Count Reducer

```
public static class Reduce extends MapReduceBase implements
    Reducer<Text,IntWritable,Text,IntWritable> {
public static void map(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));
     }
  }
```

Putting it all together

```
JobConf conf = new JobConf(WordCount.class);
conf.setJobName("wordcount");

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

conf.setMapperClass(Map.class);
conf.setCombinerClass(Reduce.class);
conf.setReducer(Reduce.class);

conf.setInputFormat(TextInputFormat.class);
Conf.setOutputFormat(TextOutputFormat.class);
FileInputFormat.setInputPaths(conf, new Path(args[0]));
FileOutputFormat.setOutputPath(conf, new Path(args[1]));
JobClient.runJob(conf);
```

Input and Output Formats

- A Map/Reduce may specify how it's input is to be read by specifying an *InputFormat* to be used
- A Map/Reduce may specify how it's output is to be written by specifying an *OutputFormat* to be used
- These default to *TextInputFormat* and *TextOutputFormat*, which process line-based text data
- Another common choice is *SequenceFileInputFormat* and *SequenceFileOutputFormat* for binary data
- These are file-based, but they are not required to be

How many Maps and Reduces

- Maps
  - Usually as many as the number of HDFS blocks being processed, this is the default
  - Else the number of maps can be specified as a hint
  - The number of maps can also be controlled by specifying the *minimum split size*
  - The actual sizes of the map inputs are computed by:
    - *max(min(block_size,data/#maps), min_split_size)*
- Reduces
  - Unless the amount of data being processed is small
    - *0.95*num_nodes*mapred.tasktracker.tasks.maximum*

Calculate number of Mappers in Hadoop

- 1) Calculate the total size of the input files by adding the size of all the files

- 2) No of Mappers = Total size calculated / Input split size defined in Hadoop configuration (*NOTE 1*)

(e.g)

- Total size calculated = 1GB (1024MB)
- Input split size = 128MB
- No of Mappers = 8 (1024 / 128)

Properties for controlling split size are

Mapreduce.input.fileinputformat.split.minsize – default value: 1 byte

Mapreduce.input.fileinputformat.split.maxsize – default value: 8192 PB (petabytes)

Dfs.blocksize – 128 MB (megabytes)

- **Splitsize = Max(Minumsize, Min(Maximumsize, Blocksize))**
- Usually minsize < blocksize < maxsize so splitsize = blocksize
- 1 byte < 128MB < 8192PB =====> Splitsize = 128MB

For example if maxsize = 64MB and blocksize = 128 MB

- Then splitsize will be limited to maxsize
- minsize < maxsize < blocksize so splitsize = maxsize
- 1 < 64MB < 128MB =====> Splitsize = 64MB

No. of Reducers

- Job.setNumreduceTasks(int) the user set the number of reducers for the job.

The right number of reducers are 0.95 or 1.75 multiplied by (<no. of nodes> * <no. of the maximum container per node>).

- With 0.95, all reducers immediately launch and start transferring map outputs as the maps finish. With 1.75, the first round of reducers is finished by the faster nodes and second trend of reducers is launched doing a much better job of load balancing.

THANK YOU