MapReduce

Definition

MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster

http://en.wikipedia.org/wiki/MapReduce

Originally published in 2004 from Google engineers Jeffrey Dean and Sanjay Ghemawat

MapReduce inspiration

The name MapReduce comes from functional programming:

map is the name of a higher-order function that applies a given function to each element of a list.

Sample in Scala:

```
val numbers = List(1,2,3,4,5)
numbers.map(x => x * x) == List(1,4,9,16,25)
```

reduce is the name of a higher-order function that analyze a recursive data structure and recombine through use of a given combining operation the results of recursively processing its constituent parts, building up a return value.

Sample in Scala:

```
val numbers = List(1,2,3,4,5)
numbers.reduce(_ + _) == 15
```

MapReduce takes an input, splits it into smaller parts, execute the code of the mapper on every part, then gives all the results to one or more reducers that merge all the results into one.

Why use MR Pattern

Map/Reduce is a pattern that is very well suited for embarrassingly parallel algorithms.

- What is an embarrassingly parallel algorithm?
  An algorithm that is very well fit to be executed multiple times in parallel.

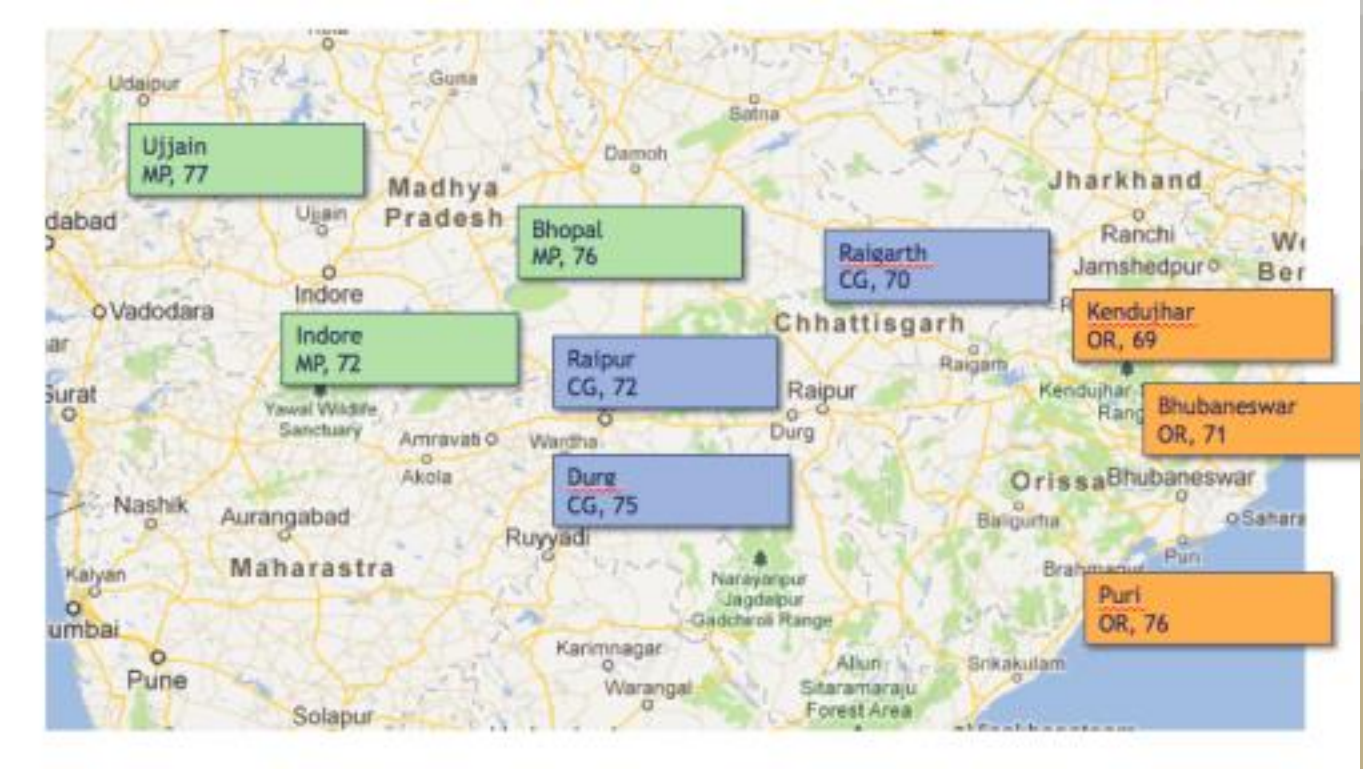
- What is very well suited for a parallel execution?
  Any algorithm that's working on data that can be isolated (remember shared nothig architecture?)

A Use-Case

- Let's say you have a list of cities, and each one has two attributes : the state it belongs to, and its yearly average temperature
- The requirement is to calculate the yearly average temperature BY STATE.

Calculate Yearly Avg Temp by state

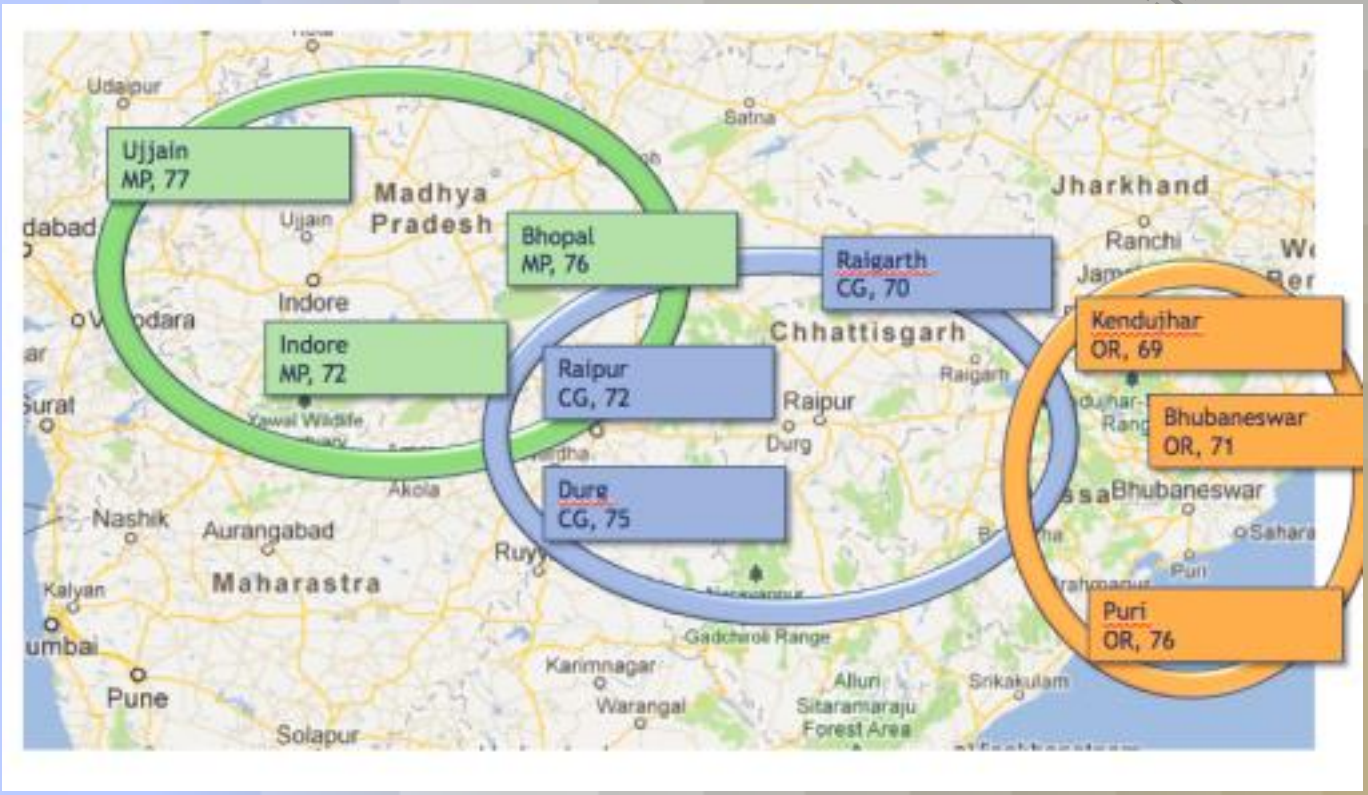
You have data by cities (not STATES), the data that was reported to you is CityName, the state to which it belongs to and the yearly average temp of that given city



1 Ujjain,MP,77
2 Indore,MP,72
3 Durg, CG,75
4 Puri, OR, 76
5 Bhub…,OR,71
6 Kendu…,OR,71
7 Raigar,CG,70
8 Raipur CG,72
9 Bhopal,MP,76

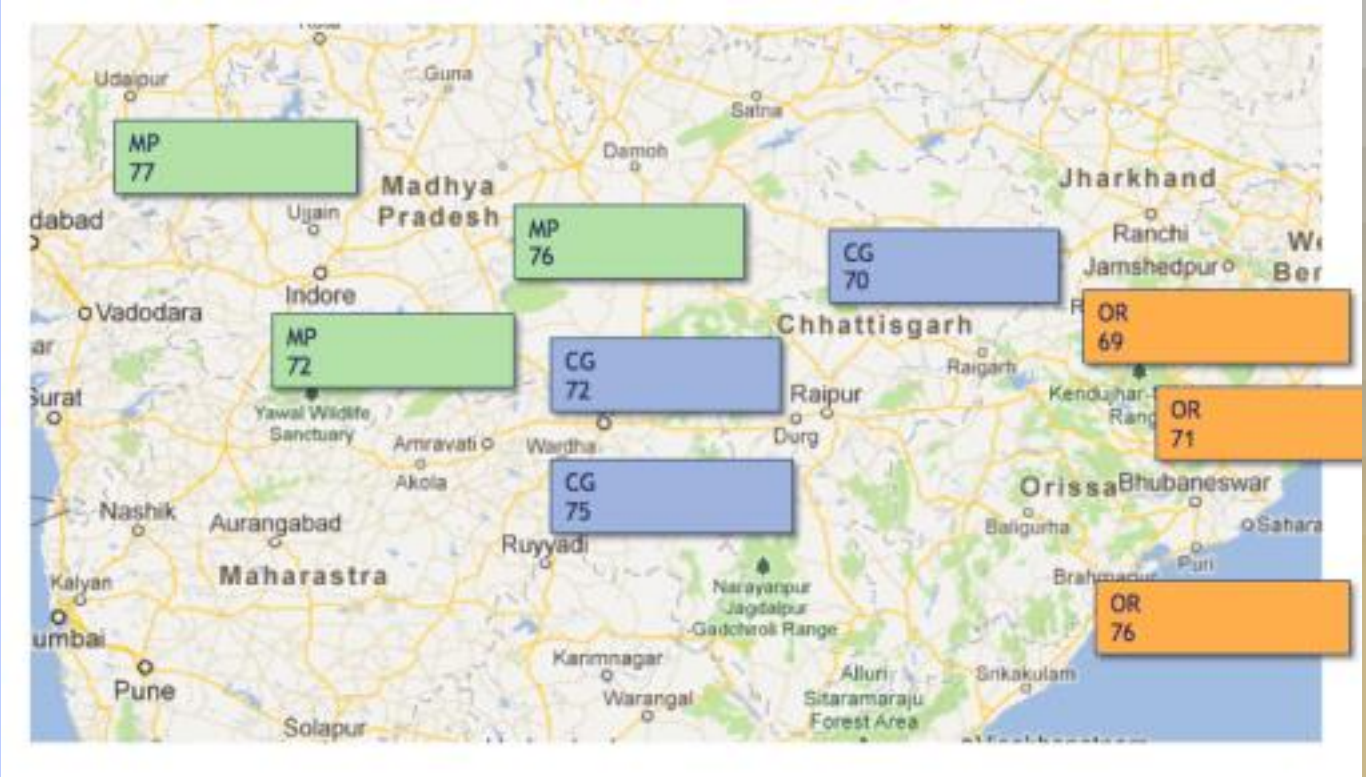
Calculate Yearly Avg Temp by state

To solve this use case, we should group the city average temperatures by state, then calculate the average of each group.



Calculate Yearly Avg Temp by state

Our usecase does not really require the city names, so we will discard those and keep only the state names and cities Temperatures. This is called **Projection**



Calculate Yearly Avg Temp by state

We now have the data that we care about, and we can regroup the temperatures values by state. We're going to get a list of temperatures averages for each state.

Calculate Yearly Avg Temp by state

We now have the data in good shape to actually do the maths… All we have to do is to calculate the average temperature for each state

We had some input data. We did a little regrouping, then we did the calculation. And all this could be executed in parallel (One parallel task for each state). In simple terms, this is MR.

Map/Reduce has 3 stages : Map/Shuffle/Reduce

Calculate Yearly Avg Temp by state

We now have the data in good shape to actually do the maths… All we have to do is to calculate the average temperature for each state

We had some input data. We did a little regrouping, then we did the calculation. And all this could be executed in parallel (One parallel task for each state). In simple terms, this is MR.

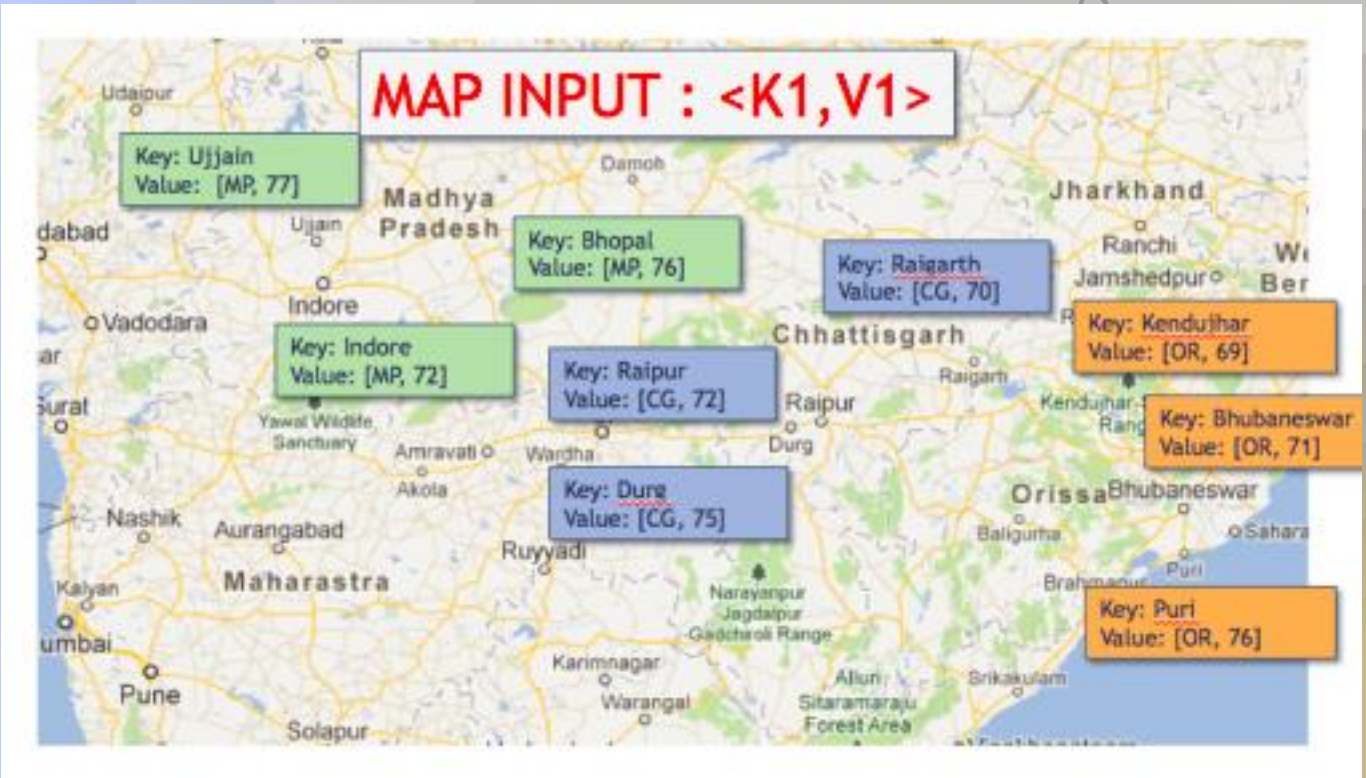
Map/Reduce has 3 stages : Map/Shuffle/Reduce

The Shuffle part is done automatically by Hadoop, you just need to implement the Map and Reduce parts.

It is the Shuffle part which takes away much of the complexities in MR pattern, and Shuffle phase is the reason behind massive scaling of Hadoop

Calculate Yearly Avg Temp by state (Hadoop Way)

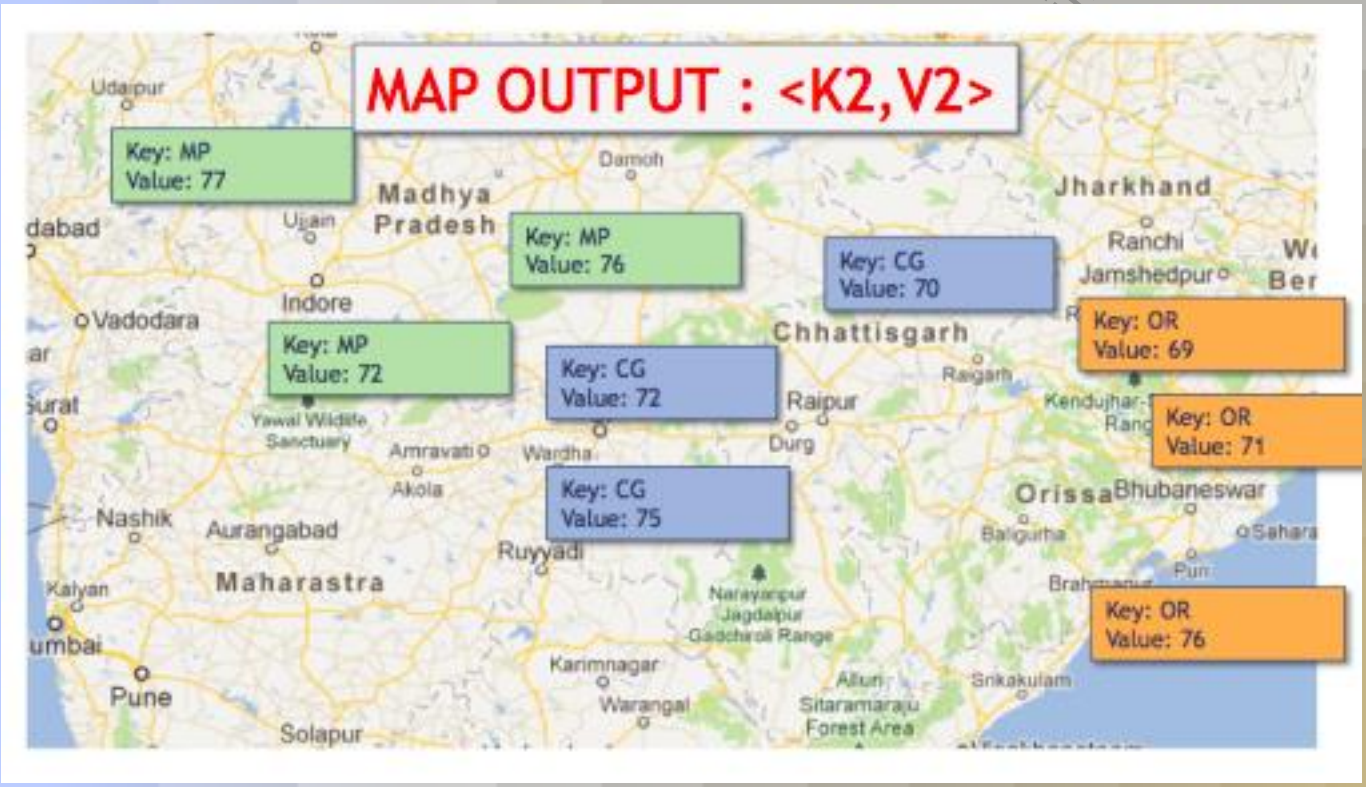
You get input data as <Key,Value> for the Map part.



1 Ujjain,MP,77
2 Indore,MP,72
3 Durg, CG,75
4 Puri, OR, 76
5 Bhub…,OR,71
6 Kendu…,OR,71
7 Raigar,CG,70
8 Raipur CG,72
9 Bhopal,MP,76

Calculate Yearly Avg Temp by state (Hadoop Way)

Since you want to regroup your temperatures by state, you're going to get rid of the city name, and the State will become the Key, while the Temperature will become the Value for the Map output.

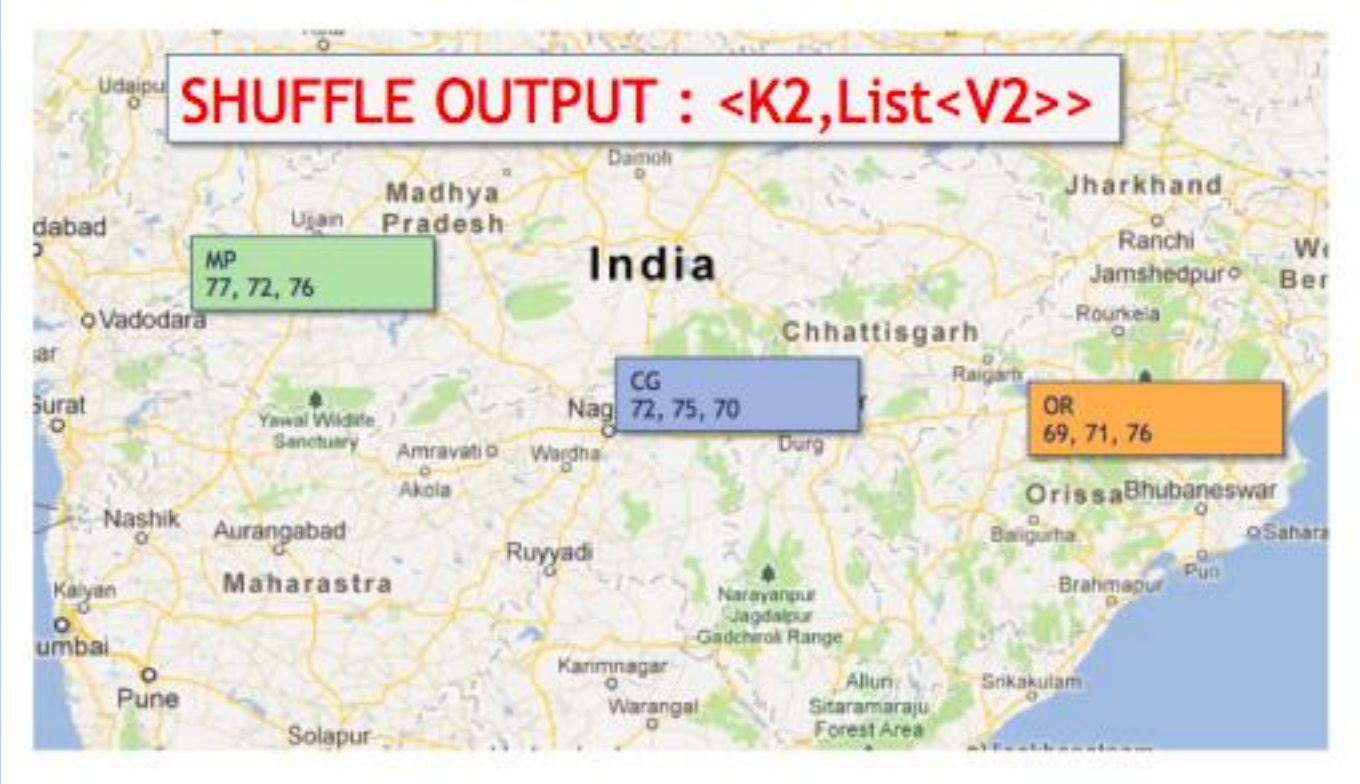


MP,77
MP,72
CG,75
OR, 76
OR,71
OR,71
CG,70
CG,72
MP,76

Calculate Yearly Avg Temp by state (Hadoop Way)

Now, the shuffle task will run on the output of the Map task. It is going to group all the values by Key, and you'll get a List<Value>

Also, the shuffle output is sorted on keys. Shuffle+Sort is taken care by Hadoop



CG,75,70,72
MP,77,72,76
OR, 76,71,71

Calculate Yearly Avg Temp by state (Hadoop Way)

Finally the Reduce task will get the input in a sorted fashion on keys as : the Key, List<Value> from the Shuffle task.

Calculate Yearly Avg Temp by state (Hadoop Way)

Reduce task is the one that does the logic on the data, in our case this is the calculation of the State yearly average temperature.

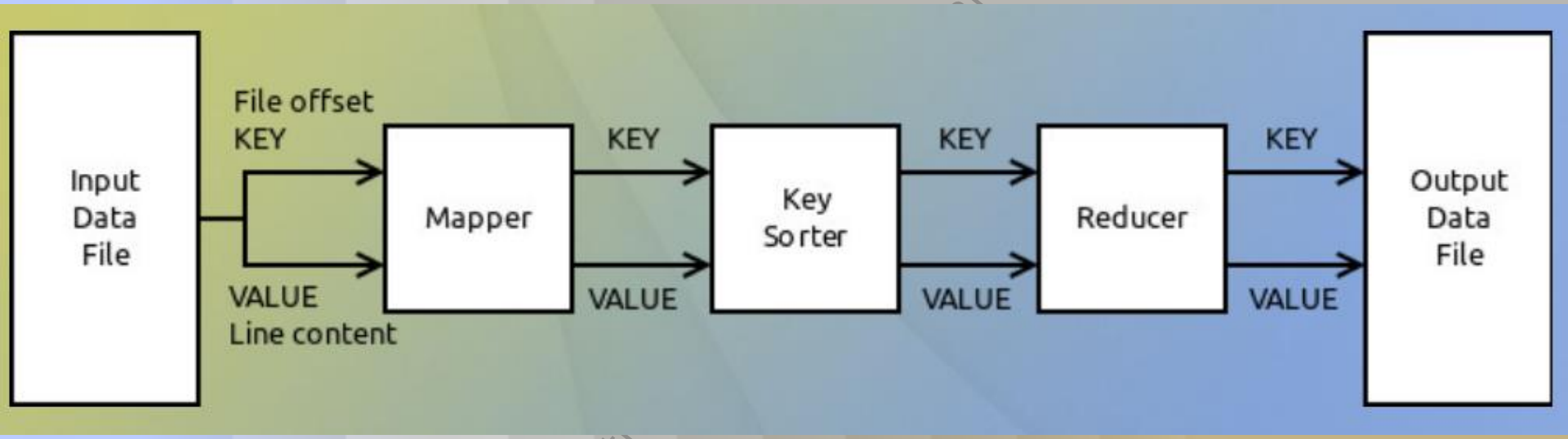
The data is shaped,operated, transported, grouped & sorted across Map/Reduce as Keys, Values:

Mapper <K1, V1> —> <K2, V2>
Reducer <K2, List<V2>> —><K3, V3>

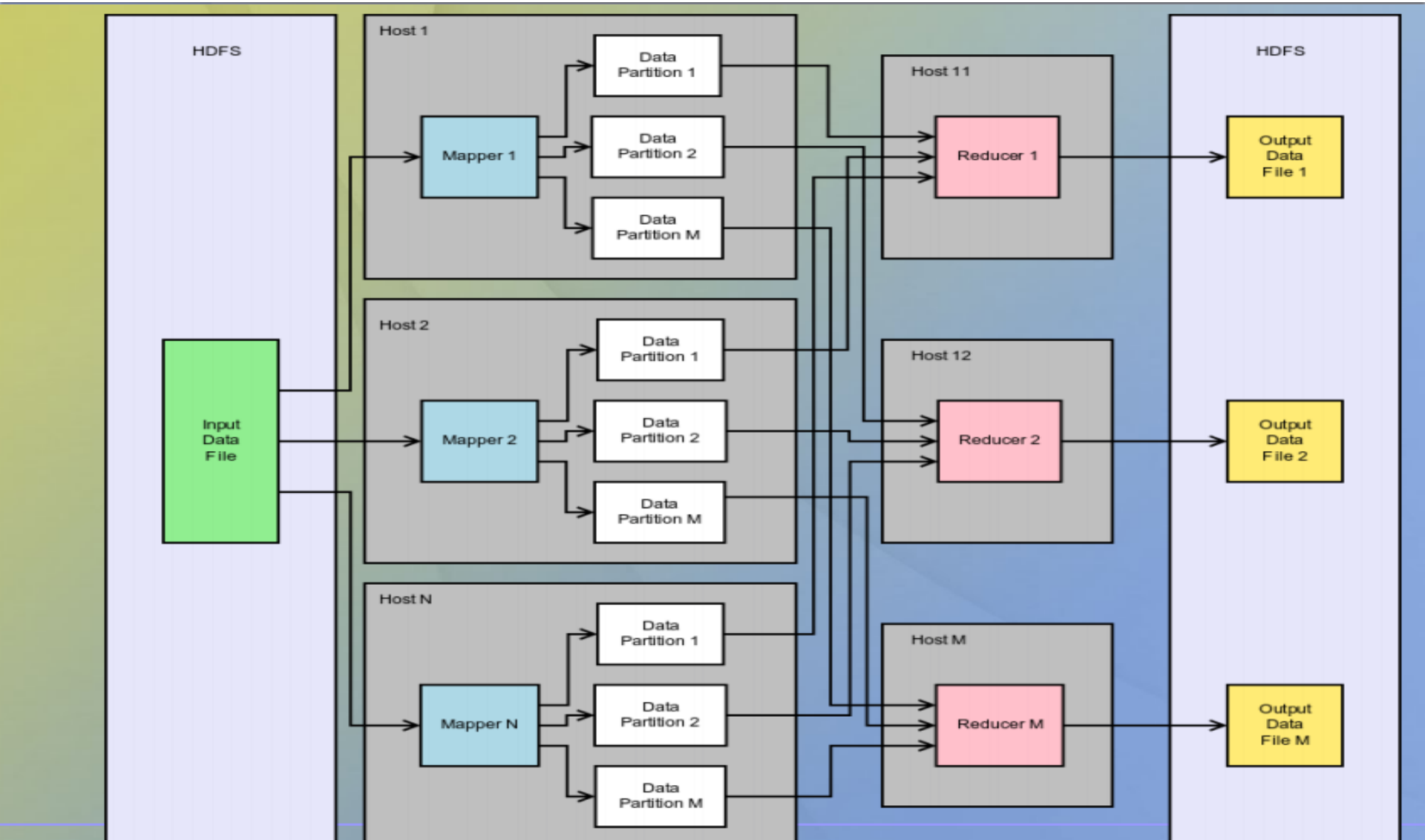
Recap

- **Init**
  - Hadoop divides the input file stored on HDFS into splits (typically of the size
  - of an HDFS block) and assigns every split to a different mapper, trying to
  - assign every split to the mapper where the split physically resides
- **Mapper**
  - locally, Hadoop reads the split of the mapper line by line
  - locally, Hadoop calls the method map() of the mapper for every line passing it as the key/value parameters
  - the mapper computes its application logic and emits other key/value pairs
- **Shuffle and sort**
  - locally, Hadoop's partitioner divides the emitted output of the mapper into partitions, each of those is sent to a different reducer
  - locally, Hadoop collects all the different partitions received from the mappers and sort them by key
- **Reducer**
  - locally, Hadoop reads the aggregated partitions line by line
  - locally, Hadoop calls the reduce() method on the reducer for every line of the input
  - the reducer computes its application logic and emits other key/value pairs
  - locally, Hadoop writes the emitted pairs output (the emitted pairs) to HDFS

Simplified flow



Overall View



Serialization

- Serializable vs Writable
  - Serializable stores the class name and the object representation to the stream; other instances of the class are referred to by an handle to the class name: this approach results in unusual bloating.
  - The deserialization process creates a new instance of the object, while Hadoop needs to reuse objects to minimize computation.
  - Hadoop wanted a serialization mechanisn which is compact, fast, extensible and Interoprable and hence -
- Hadoop introduced the two interfaces Writable and WritableComparable that solve these problem

Writable Wrappers

Java primitive	Writable implementation
boolean	BooleanWritable
byte	ByteWritable
short	ShortWritable
int	IntWritable VIntWritable
float	FloatWritable
long	LongWritable VLongWritable
double	DoubleWritable

Java class	Writable implementation
String	Text
byte[]	BytesWritable
Object	ObjectWritable
null	NullWritable

Java collection	Writable implementation
array	ArrayWritable ArrayPrimitiveWritable TwoDArrayWritable
Map	MapWritable
SortedMap	SortedMapWritable
enum	EnumSetWritable

Example Custom Writable

```
public class EmployeeWritable implements WritableComparable {
       // Some data
       private int empNo;
       private long salary;

       public void write(DataOutput out) throws IOException {
         out.writeInt(empNo);
         out.writeLong(salary);
       }

       public void readFields(DataInput in) throws IOException {
         empNo = in.readInt();
         salary = in.readLong();
       }

       public int compareTo(EmployeeWritable o) {
         int thisValue = this.empNo;
         int thatValue = o.empNo;
         return (thisValue < thatValue ? -1 : (thisValue==thatValue ? 0 : 1));
       }

       public int hashCode() {
         final int prime = 31;
         int result = 1;
         result = prime * result + empNo;
         result = prime * result + (int) (salary ^ (salary >>> 32));
         return result
       }
     }
```

Common Terms

Term	Meaning
Job	The whole process to execute: the input data, the mapper and reducers execution and the output data
Task	Every job is divided among the several mappers and reducers; a task is the job portion that goes to every single mapper and reducer
Split	The input file is split into several splits (the suggested size is the HDFS block size, 64Mb) 128MB for 2.x
Record	The split is read from mapper by default a line at the time: each line is a record. Using a class extending `FileInputFormat`, the record can be composed by more than one line
Partition	The set of all the key-value pairs that will be sent to a single reducer. The default partitioner uses an hash function on the key to determine to which reducer send the data

An example – the data

Ujjain,MP,77
Bhopal,MP,76
Indore,MP,72
Raipur,CG,72
Durg,CG,75
Raigarth,CG,70
Kendujhar,OR,69
Bhubaneswar,OR,71
Puri,OR,76

Put this file in HDFS into /temperatures/input.txt, later in the code we would tell Hadoop to pass this file content to the Mappers

Code – the Mapper, Reducer

```
public class TMapper extends Mapper<Text, Text, Text, IntWritable> {
        public void map(Text key, Text value, Context context) throws IOException, InterruptedException {
          String[] data = value.toString().split(",");
          String state = data[1];
          IntWritable temp = new IntWritable(data[2]);
          context.write(new Text(state),temp);
         }
}


public class TReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
 public void reduce(final Text key, final Iterable<IntWritable> values, final Context context) throws IOException,
InterruptedException {
   int sumOfTemperatures = 0;
   int numValues = 0;

 for (IntWritable temperature : values) {
    sumOfTemperatures += temperature.get();
    numValues++;
  }
   int average = sumOfTemperatures / numValues;
   context.write(key, new IntWritable(average));
 }
}
```

Code – the Driver

```
public class TDriver{
  public static void main(String args[]) throws Exception{
   Configuration conf = new Configuration();
   Job job = new Job(conf, "AvgTemperatureJob");
   String outputDir = "OUTPUT_" + System.currentTimeMillis();

   FileInputFormat.addInputPath(job, new Path(inputFile));
   job.setInputFormatClass(TextInputFormat.class);


   FileOutputFormat.setOutputPath(job, new Path(outputDir));
   job.setOutputFormatClass(TextOutputFormat.class);

   job.setJarByClass(TDriver.class);
   job.setMapperClass(TMapper.class);
   job.setReducerClass(TReducer.class);

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


   job.waitForCompletion(true);
   System.out.println("Job Completed, results are stored in the directory - " + outputDir);

  }
}
```

Configuration

- Add the below content to mapred-site.xml in the etc/hadoop dir.

```
<configuration>
            <property>
             <name>mapreduce.framework.name</name>
              <value>yarn</value>  <!-- or set this to local for debug needs -->
            </property>
</configuration>
```

- Add the below content to yarn-site.xml in the etc/hadoop dir.

```
<configuration>
        <property>
         <name>yarn.nodemanager.delete.debug-delay-sec</name>
         <value>600</value>
         </property>
        <property>
            <name>yarn.resourcemanager.hostname</name>
            <value>localhost</value>
        </property>
        <property>
            <name>yarn.nodemanager.aux-services</name>
            <value>mapreduce_shuffle</value>
        </property>
        <property>
            <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
```

Compile the code

- Using old way, add jars to the IDE and compile, generate a jar out of compiled artifacts – not recommended
- Use Maven, we would use this going forward.
- The pom.xml (with dependencies and required plugins) is as below, or refer to code that is checked into github


```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
```

Run the code

Copy generated jar to your sandbox

Start Yarn

start-yarn.sh/start-yarn.cmd

Run using Yarn

yarn jar theJarName.jar thePackage.Tdriver

Assignment

- Pick real time data for All India sasonal Annual Min/Max temperatures series from 1901 – 2014) from below link https://data.gov.in/catalog/all-india-seasonal-and-annual-minmax-temperature-series
- The data is layed out year wise, with min and max, but not averages.
- Compute average temperature year wise and spit data in the format of Year, Average.
- We will use this result for some analytics.