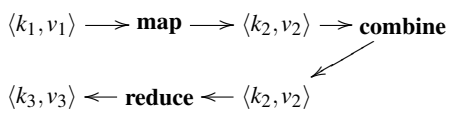
Hadoop MapReduce Framework

Overview A MapReduce *job* splits the input dataset into independent chunks processed by *map* tasks in completely parallel manner, followe by *reduce* tasks which aggregates their output. Typically both the input and the output of the job are stored in a Hadoop `FileSystem`.

MapReduce inputs and outputs The MapReduce framework operates exclusively on ⟨*key, value*⟩ pairs. The *key*s and *value*s have to be serializable as `Writable`s and additionally the *key*s have to be *WritableComparable*s in order to facilitate grouping by the framework.

The following is an overall flow of a MapReduce Job:



Hadoop mapper The mapper takes input ⟨*key, value*⟩ pairs, perform computation, and generates a set of intermediate ⟨*key, value*⟩ pairs.

```
map(KEYIN key,Iterable<VALUEIN> values,
    Mapper.Context ctx);

public class WordCount {
  public static class Map
    extends Mapper<LongWritable, Text,
                   Text, IntWritable> {
    public void map(LongWritable key,
                    Text value,
                    Context context) {
      String line = value.toString();
      StringTokenizer tk = new ...(line);
      while (tk.hasMoreTokens()) {
        word.set(tk.nextToken());
        context.write(word, one);
      }
    }
  }
}
```

Hadoop reducer The Hadoop reducer reduces a set of intermediate values *which share a key* to a (usually smaller) set of values.

A reducer has three primary phases: *shuffle*, *sort*, and *reduce*. In the **shuffle** phase, input to the `Reducer` is sorted output of the mappers. In this phase, the framework fetches the relevant partition of the output of all the mappers, via HTTP. In the **sort** phase, the framework groups `Reducer` inputs by keys in this phase (since different mappers may have output the same key). At the **reduce** phase, the `reduce()` method is called for each ⟨*key, list of values*⟩ pair in the grouped inputs.

```
reduce(KEYIN key,Iterable<VALUEIN> values,
       Reducer.Context ctx);

public class WordCount {
  public static class Reduce
    extends Reducer<Text, IntWritable,
                    Text, IntWritable> {
    public void map(Text key,
                    Iterable<IntWritable> values,
                    Context context) {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      context.write(key, new IntWritable(sum));
    }
  }
}
```

The heart of `Reducer` is its `reduce()` method. This method is called *once per key*. The second argument is an `Iterable/` which returns all the values associated with the key.

Running MapReduce jobs

```
public class WordCount {
  public int run(String[] args) {
    Job job = new Job(getConf());
    job.setJarByClass(WordCount.class);

    job.setOutKeyClass(Text.class);
    job.setOutValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat);
    job.setOutputFormatClass(TextOutputFormat);
  }

  public static void main(String[] args) {
    ToolRunner.run(new WordCount(), args);
  }
}
```

How Hadoop runs a MapReduce job For a single MapReduce job, there four independent entities: *clinent*, *jobtracker*, *tasktracker*, and *HDFS*.