50002 - Software Engineering Design - Lecture 10

Oliver Killane

14/11/21

Lecture Recording

Lecture recording is available here

MapReduce

A system built by Google to work on very large datasets, allowing data processing tasks to be spread over many computers in a farm.
The main difficulties win distributed computing considered were:

- If many projects are changed to be distributed, then lots of code must be changed, much will be duplicated & there is potential for bugs.
- Ordinging interactions and sharing data across many networked computers is difficult.
- Probability of a hardware failure grows with the amount of hardware.

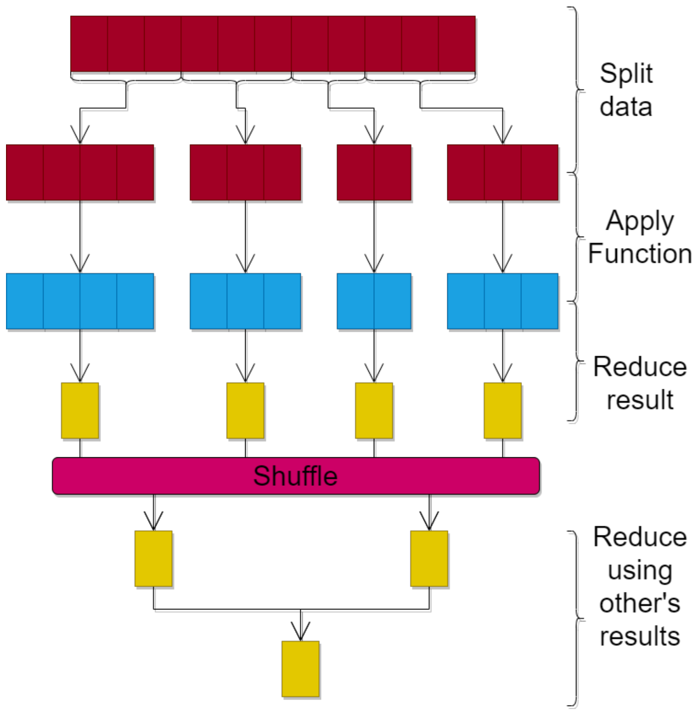
MapReduce takes tasks written by programmers, and distributises it across a number of systems, handling failures & can be easily used through an **API**. Furthermore any improvements to the system will immediately be reflected in all projects using it.

Google wanted to index the world wide web, this is a large problem that is best suited to highly distributed programming.

```
// Calculate the sum of the squares of numbers in a list.

var square = function (x) { return x * x }
var sum = function (x,y) { return x + y }

[1,2,3,4,5].map(square).reduce(sum);
```



The shuffle stage is important, to increase performance we want to transmit/copy the minimum amount of data between computers.

For **MapReduce** the signatures are different to lists & integers:

$$map : (k_1, v_1) \to list(k_2, v_2)$$

$$reduce : list(k_2, list(v_2)) \to list(k_2, v_2)$$

For example: We have k_1 = line number and v_1 = line of text

Key	**Value**
1	I think that it's extraordinarily important that we in computer science keep fun in computing.
2	When it started out, it was an awful lot of fun. Of course, the paying customers got shafted every
3	now and then, and after a while we began to take their complaints seriously. We began to feel as
4	if we really were responsible for the successful, error-free perfect use of these machines. I don't
5	think we are. I think we're responsible for stretching them, setting them off in new directions, and
6	keeping fun in the house. I hope the field of computer science never loses its sense of fun

$map(k_1, v_1) \to [(k_2, v_2)]$ where k_2 = word and $v_2 = 1$.

```
[("I",1),("think",1),("that",1),("it's",1),("extraordinarily",1),("important",1),("
↪ that",1),("we",1),("in",1),("computer",1),("science",1),("keep",1),("fun",1),(
↪ "in",1),("computing.",1)]
```

The we reduce with + for each key:
$reduce[(k_2, v_2)] \to [(k_2, [v_2])]$:

```
[('I', 1),( 'think', 1),( 'that', 2),( "it's", 1),( 'extraordinarily', 1),( '
    ↪ important', 1),( 'we', 1),( 'in', 2),( 'computer', 1),( 'science', 1),( 'keep'
    ↪ , 1),( 'fun', 1),( 'computing.', 1)]
```

Alternatives

Other projects have been developed to perform a similar role to **MapReduce** such as **Hadoop**.

Java Example

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

    public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {

      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
         word.set(tokenizer.nextToken());
         context.write(word, one);
      }
   }
}


public class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
                                      throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}


public class WordCount {
    public static void main(String[] args) throws Exception {
       Configuration conf = new Configuration();

       Job job = new Job(conf, "wordcount");

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

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

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

```

```java
        FileInputFormat.addInputPath(job, new Path("some path"));
        FileOutputFormat.setOutputPath(job, new Path("some path"));

        job.waitForCompletion(true);
    }
}
```