Lab 4 - WordCount

BI12-149 - Nguyen Duc Phuc Tuong

April 2024

1 System Architecture

We will use Docker to host a Hadoop cluster, which consists of:

- 1 master node containing NameNode and ResourceManager
- 3 slave nodes containing NodeManager, DataNode and our MapReduce computing model.

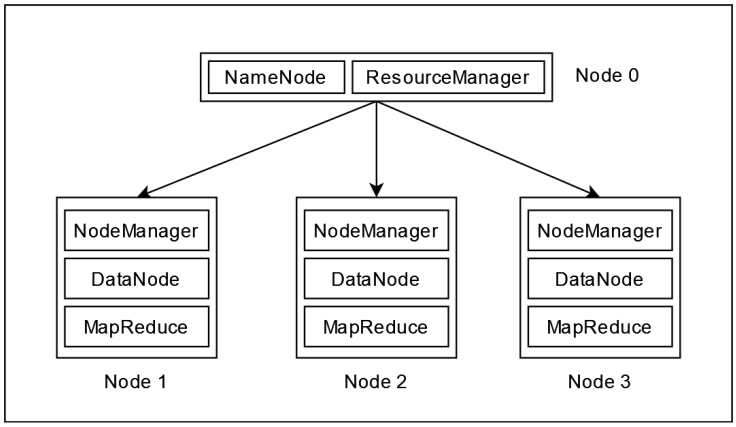

Figure 1: System Architecture

2 Implementation

2.1 Mapper

The Mapper class takes input data and splits it into tokens (words), then generates key-value pairs for each token. Each word serves as the key, while the value is set to '1', indicating the occurrence of that word. This process iterates through all tokens, effectively creating a map where words are associated with their frequencies.

```
public static class Mapper 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);
              }
        }
    }
```

2.2 Reducer

The Reducer class receives keys and their associated values from the Mapper, sums up these counts, and outputs the total count for each word to the context. The output consists of key-value pairs where the key represents a word, and the value represents the total count of occurrences for that word. This process repeats for all unique words in the input data.

```
public static class Reducer 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);
    }
}
```

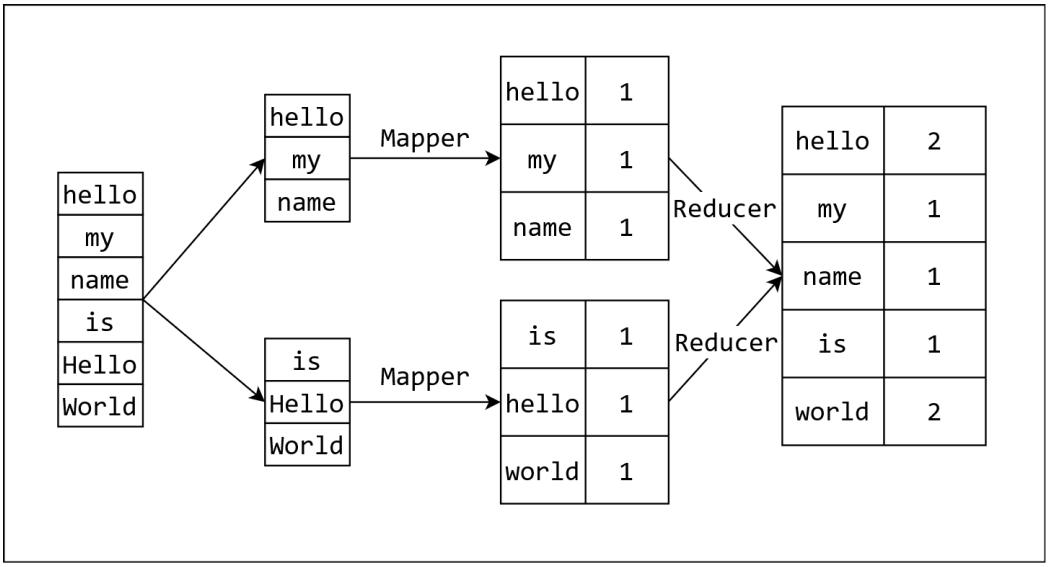

Figure 2: MapReduce algorithm