Lab 4 - WordCount

BI12-149 - Kieu Huy Hai

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.

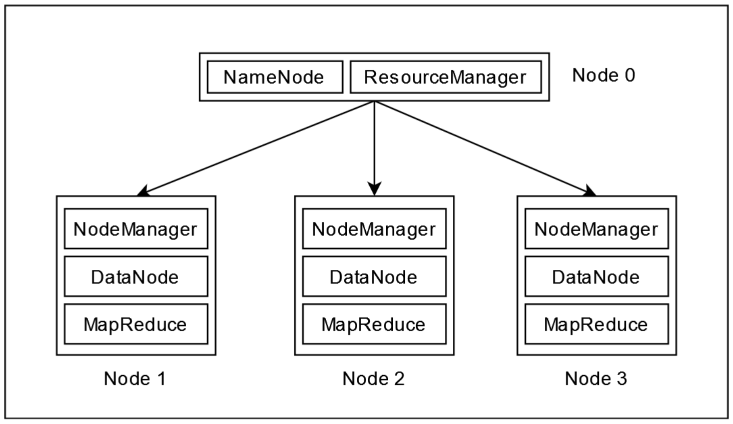


Figure 1: System Architecture

Hadoop is a robust framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale from single servers to thousands of machines, each offering local computation and storage. This makes Hadoop particularly suitable for MapReduce, a computational model that simplifies large-scale data processing across distributed environments.

2 Implementation

2.1 `Mapper`

The `Mapper` takes data as input and breaks it into tokens (words, in this case). For each token, it writes a key-value pair to the context. The key is the token, and the value is '1'. This process is repeated for all tokens, effectively creating a frequency map where each word is associated with the number of times it appears in the input data.

```java
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` takes a key and a set of associated values from the Map function as input, sums up these counts, and writes the total count for each word to the context. The output is a set of key-value pairs where the key is a word, and the value is the total count of that word in the data. This process is repeated for all unique words in the input data.

```java
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);
    }
}
```

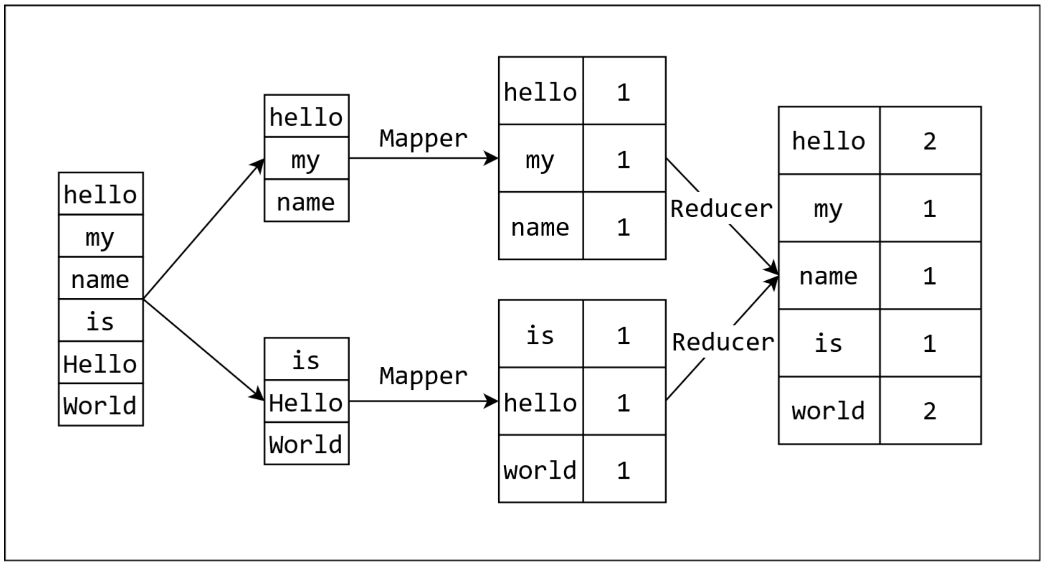
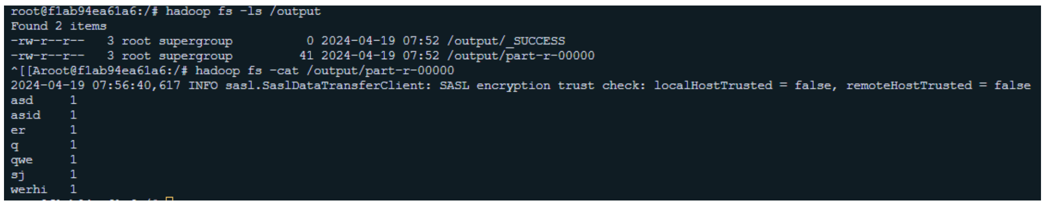


Figure 2: `MapReduce` algorithm

3 Demo



```
root@f1ab94ea61a6:/# hadoop fs -ls /output
Found 2 items
-rw-r--r--   3 root supergroup          0 2024-04-19 07:52 /output/_SUCCESS
-rw-r--r--   3 root supergroup         41 2024-04-19 07:52 /output/part-r-00000
^[[Aroot@f1ab94ea61a6:/# hadoop fs -cat /output/part-r-00000
2024-04-19 07:56:40,617 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
asd     1
asid    1
er      1
q       1
qwe     1
sj      1
werhi   1
```

Figure 3: Demo