Exercise Session 7 : MapReduce

Agenda :

① MapReduce as a System

* Motivation and design

② MapReduce as a Computational Model

* Example questions

MapReduce : Motivation

Eg, all YouTube videos with their metadata.

① PBs of data ⇒ Distributed file system (HDFS)

* Global view ⇒ Location of every piece of data

  Global view → NameNode

  piece of data → Replica

② Minimize data movement ⇒ Data locality

Why? I/O kills performance

* Centralized scheduling ⇒ Colocating compute with data

  Centralized scheduling → JobTracker

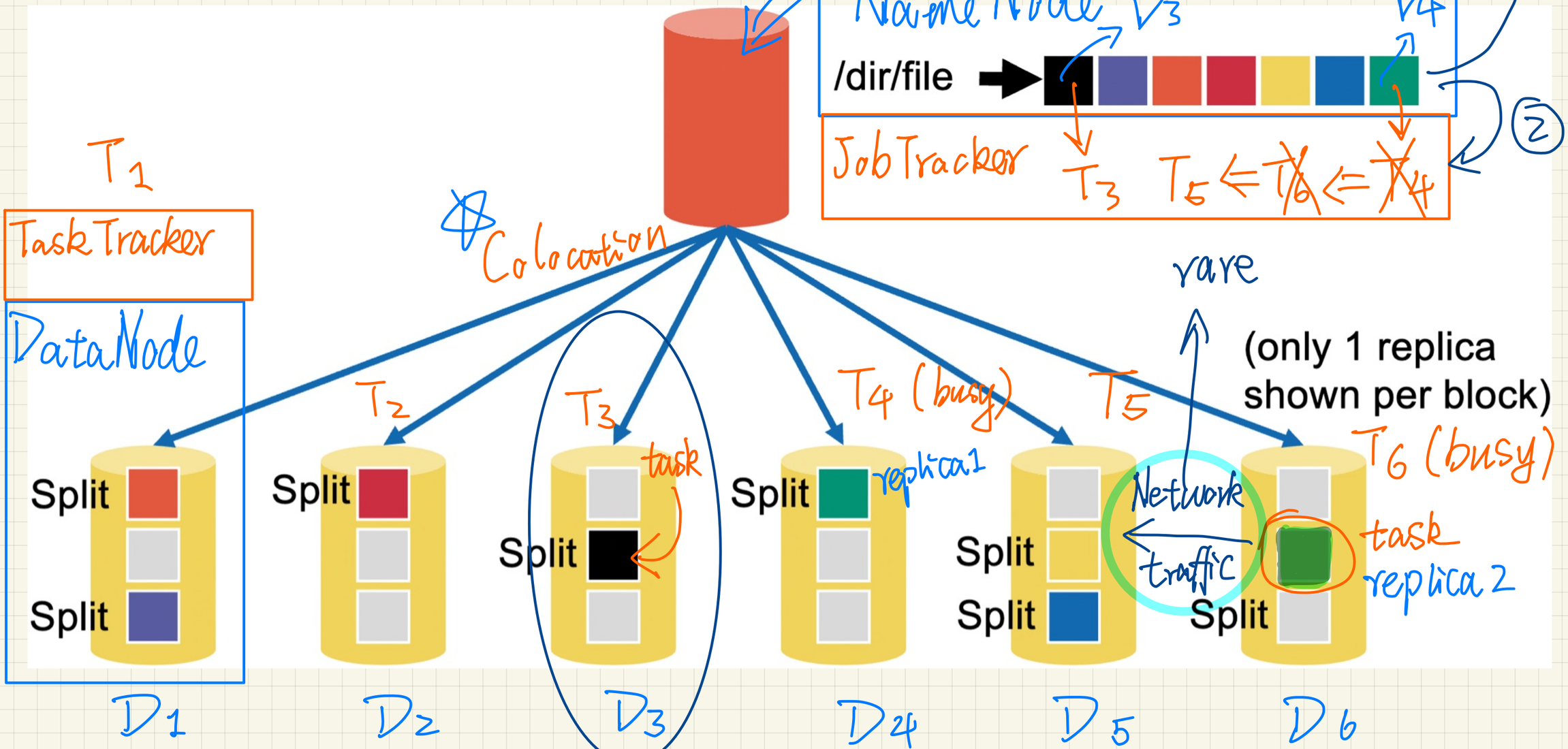
  compute → Map/Reduce tasks

  data → Splits

MapReduce : Design

① Data replication

② Task scheduling



MapReduce as a Computational Model

① (Many) analytical queries ⇒ 2 phases: mapping and reducing

⇒ Paper [OSDI '04]

② Map/Reduce tasks are embarrassingly parallel

⇒ Divide & Conquer

Example

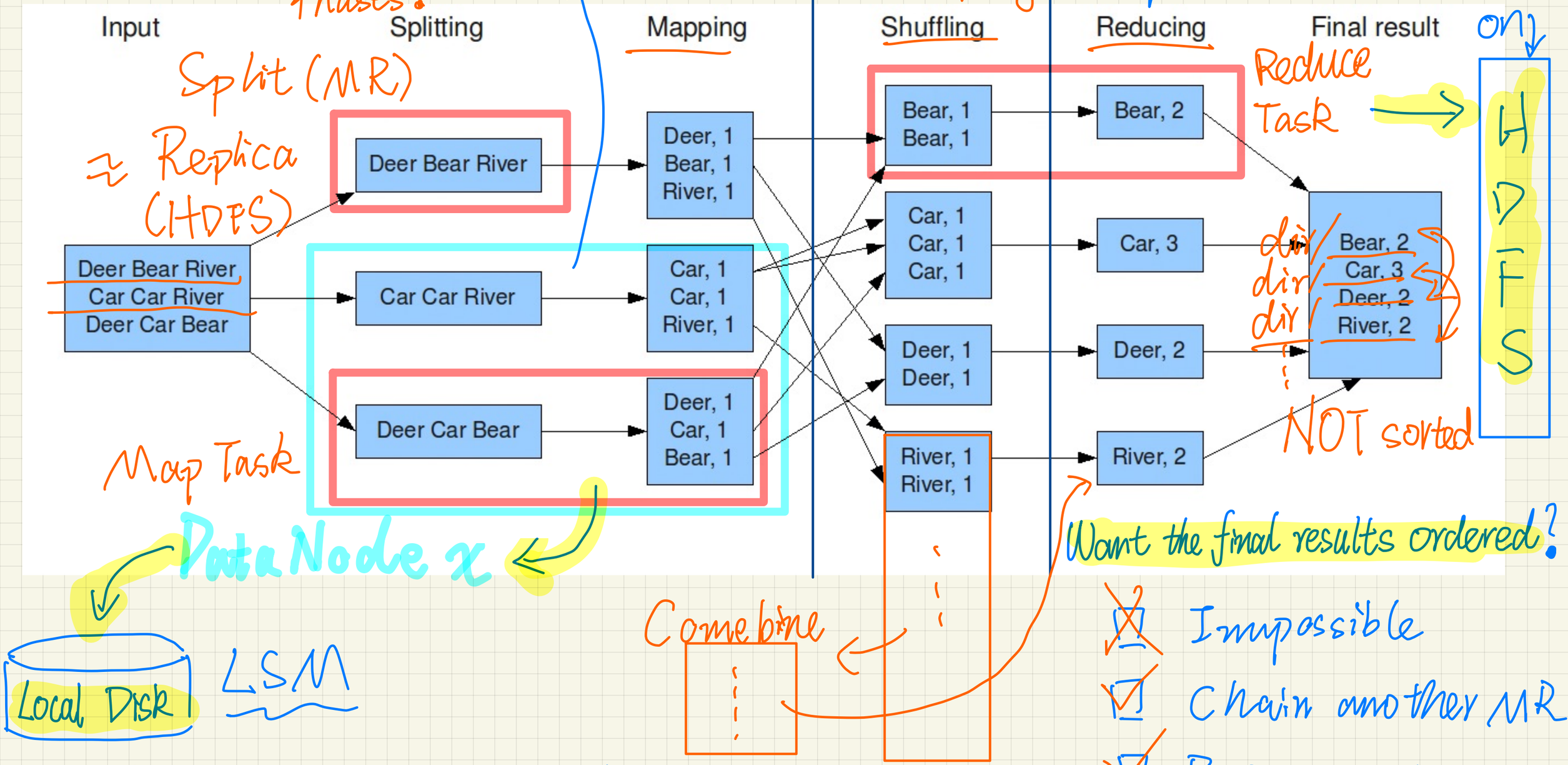
Map task failed after completion? ⇒ Re-execution

① Sorting
② Grouping

https://www.todaysoftmag.com/article/1358/hadoop-mapreduce-deep-diving-and-tuning

Reduce task failed after completion ? ⇒ Don't care

Phases:



Split (MR) ≈ Replica (HDFS)

Reduce Task

dir
dir
dir

NOT sorted

Map Task

DataNode x

Local Disk

LSM

Combine

Want the final results ordered?

- ☒ Impossible
- ☑ Chain another MR
- ☑ Post processing
- ☑ Single reducer

Stored in ASC order of keys ⇒ Dremel: In-memory (faster) shuffling

Typical Questions

5. Reverse engineering

Conceptually, a map function takes an input a key-value pair and emits a list of key-values pairs, while a reduce function takes in input a key with an associated list of values and returns a list of values or key-value pairs. Often the type of the final key and value is the same of the type of the intermediate data:

- map `(k1,v1) --> list(k2,v2)`
- reduce `(k2,list(v2))--> list(k2, v2)`

Analyze the following Map and Reduce functions, written in pseudo-code, and answer the questions below.

```
function map(key, value)
  emit(key, value);

function reduce(key, values[])
  z = 0.0
  for value in values:
    z += value
  emit(key, z / values.length())
```

Implementation ⇒ functionality

Questions

1. Explain what is the result of running this job on a list of pairs with type ([string], [float]).
2. Write the equivalent SQL query.
3. Could you use this reduce function as combine function as well? Why or why not?
4. If your answer to the previous question was *yes*, does the number of different keys influences the effectiveness of the combine function? If you answer was *no*, can you change the map and reduce functions in such a way that the new reduce function can be used as combine function?

6. True or False

Say if the following statements are *true* or *false*, and explain why.

1. Each map function must generate the same number of key/value pairs as its input had.
2. The TaskTracker is responsible for scheduling map functions and reduce functions and make sure all nodes are correctly running.
3. The input key/value pairs of map functions are sorted by the key.
4. MapReduce splits might not correspond to HDFS block.
5. One single Reduce function is applied to all values associated with the same key.
6. Multiple Reduce functions can be assigned pairs with the same value.
7. In Hadoop MapReduce, the key-value pairs a Reduce function outputs must be of the same type as its input pairs.

7. Some more MapReduce and SQL

Design, in Python or pseudo-code, MapReduce functions that take a very large file of integers and produce as output:

1. The largest integer.
2. The average of all the integers.
3. The set of integers, but with each integer appearing only once.
4. The number of times each unique integer appears.
5. The number of distinct integers in the input.

Tips: (I) Single-value result ⇒ Map everthing to the same key

(II) Multiple values / statistics

For each of these, write the equivalent SQL query, assuming you have a column `values` that stores all the integers.

Solution

1. The largest integer.

```
def map_(line, val):
    return ('', val)

def reduce_(key, values):
    return ('', max(values))

SELECT MAX(values)
...
```

2. The average of all the integers.

```
def map_(line, val):
    return ('', (1, val))

def reduce_(key, values):
    vals = [v for value[1] in values]
    return ('', sum(vas) / len(values))

SELECT AVG(values)
...
```

3. The set of integers, but with each integer appearing only once.

```
def map_(line, val):
    return (val, '')

def reduce_(key, values):
    return ('', values[0])

SELECT DISTINCT(values)
...
```

4. The number of times each unique integer appears.

```
def map_(line, val):
    return (val, 1)

def reduce_(key, values):
    return (key, sum(values))

SELECT values, COUNT (*) ... GROUP BY values
...
```

5. The number of distinct integers in the input.

```
def map_(line, val):
    return ('', val)

def reduce_(key, values):
    return ('', len(set(values))
#* OR just reuse 3. and add a postprocessing step.

SELECT COUNT (DISTINCT values)
...
```

Try It Out !

Big Data – Exercises – Solution

Fall 2023 – Week 7 – ETH Zurich

MapReduce

This exercise consists:

- Architecture and theory of MapReduce

Important!

If you are using a intel machine please work in this folder. If you are using Mac silicon (ARM architecture, M1, M2, M3 etc.) please go to ./exercise07-aarch64, and work in that folder.

1. Setup a cluster

Create an Hadoop cluster

Start the Hadoop cluster (in pseudo-distributed mode), similar to the HDFS exercise session, by running:

```
sudo docker-compose up
```

Wait for a couple minutes until the terminal no longer outputs startup logs.

2. Write a Word Count MapReduce job

We want to find which are the most frequently-used English words. To answer this question, we prepared a big text files (1.7GB) where we concatenated thousands of books of the Gut a smaller text file `gutenberg_x0.1.txt` is used.

2.1 Load the dataset

The dataset we provide consists of a concatenation of thousands of books (`gutenberg.txt`). However we provide 2 versions, click the file below to download from Polybox.

- Log in into the NameNode container by launching a shell on it:

`docker exec -it namenode /bin/bash`

- Load the dataset into the HDFS filesystem:

  With `ls -lh` you should see the 2 files mentioned above. These files are now in the "local" (remember, we are in containers) hard drive of your NmaeNode.

  Upload the files into HDFS where they can be consumed by MapReduce:

`hdfs dfs -copyFromLocal *.txt /`

2.2 Understand the MapReduce Java API

We wrote a template project that you can use to experiment with MapReduce named `mapreduce.zip`. Store **on your local machine, not in the container** and unzip the following pa

Note: Before you docker cp, make sure you remove the old mapreduce directory from the NameNode

`unzip mapreduce.zip`
`docker cp mapreduce namenode:/mapreduce`
Now examine the content of the the `./mapreduce/src` folder. You will see one Java class:

- *MapReduceWordCount*: a skeleton for a MapReduce job that loads data from file

Start looking at *MapReduceWordCount*. You can see that the *main* method is already provided. Our `WordCountMapper` and `WordCountReducer` are implemented as classes that e `Reducer`. For this exercise, you only need to consider (and override) the `map()` method for the mapper and the `reduce()` method for the reducer.

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {

    protected void map(KEYIN key, VALUEIN value, Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>.Context context) {
        context.write(key, value);
    }
}
public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    protected void reduce(KEYIN key, Iterable<VALUEIN> values, Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>.Context context) {
        Iterator var4 = values.iterator();

        while(var4.hasNext()) {
            Object value = var4.next();
            context.write(key, value);
        }

    }
```

Solution

1. Since the output of mapper and reducer must be serializable, they have to implement the Writable interface. Additionally, the key classes have to implement the WritableComparab by the framework. Hadoop provides classes implementing these interfaces for string, boolean, integers, long and short values.
2. Mapper and Reducers are identity functions. Since there is always a shuffling phase, the overall job performs sorting by input key.
3. Context allows the Mapper/Reducer to interact with the rest of the Hadoop system. It includes configuration data for the job and the methods for mapper and reducer to emit outp also use the Context to set parameters to mappers and reducers runnning on different nodes.

2.3 Write and run your MapReduce wordcount

Edit the provided skeleton and implement mapper and reducer to implement a word count. The goal is to know how many times each unique word appears in the dataset. You can cons characters separated by a whitespace, or implement a more sophisticated tokenizer if you wish.

- Can you use your Reducer as Combiner? If so enable it by uncommenting the appropriate line in the `main` method.

Once you are confident on your solution you can transfer it back to the container, compile it and run it, from the **mapreduce/src** folder.

```
javac *.java -cp $(hadoop classpath)
jar cvf MapReduceWordCount.jar *.class
```

Inside the folder, you will now find `MapReduceWordCount.jar`. Run the map reduce job on the cluster using:

```
yarn jar MapReduceWordCount.jar MapReduceWordCount /gutenberg_x0.1.txt /tmp/results
```

To get the results after the job is done, simply copy to the local directory with the resuts:

```
hdfs dfs -copyToLocal /tmp/results
```

The process is very similar to the one for HBase of last week. Answer the following questions:

1. Run the MapReduce job on the cluster with the default configuration and 4 DataNodes using only `gutenberg_x0.1.txt` for now. *(Note: if you want to run your job again, you fi result folder because Hadoop refuses to write in the same location)*:

   ```
   hdfs dfs -rm -r <path-to-hdfs-output-folder>
   ```

2. How many map and reduce tasks were created with the default configuration?
3. Does it go faster with more reduce tasks? Uncomment and experiment with `job.setNumReduceTasks()`. What is the disadvantage of having multiple reducers? (Hint: check th

```
private static class WordCountMapper 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);
            }
        }
    }
```

Reducer

```
private static class WordCountReducer 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);
        }
    }
```

1. Some figures are provided below.

2. In my case, the number of map tasks was 2(or 3) and the number of reduce tasks 1. By default, the number of map tasks for a given job is driven by the number of input splits. The `mapred.map.tasks` can be used to set a *suggested* value. For reduce tasks, 1 is the default number. This might be a bad choice in terms of performance, but on the other hand without further merging necessary.

3. Generally, one reducer might become a bottleneck. In this case, the bottleneck is more evident if no combiner is used. However, using multiple reducers, multiple output files will b merging step is necessary to get the same result. Note: according to the official documentation, the right number of reduces is 0.95 or 1.75 multiplied by [no. of nodes] * [no. of m With 0.95 all of the reduces can launch immediately and start transferring map outputs as the maps finish. With 1.75 the faster nodes will finish their first round of reduces and lau doing a much better job of load balancing. The scaling factors above are slightly less than whole numbers to reserve a few reduce slots in the framework for speculative-tasks and

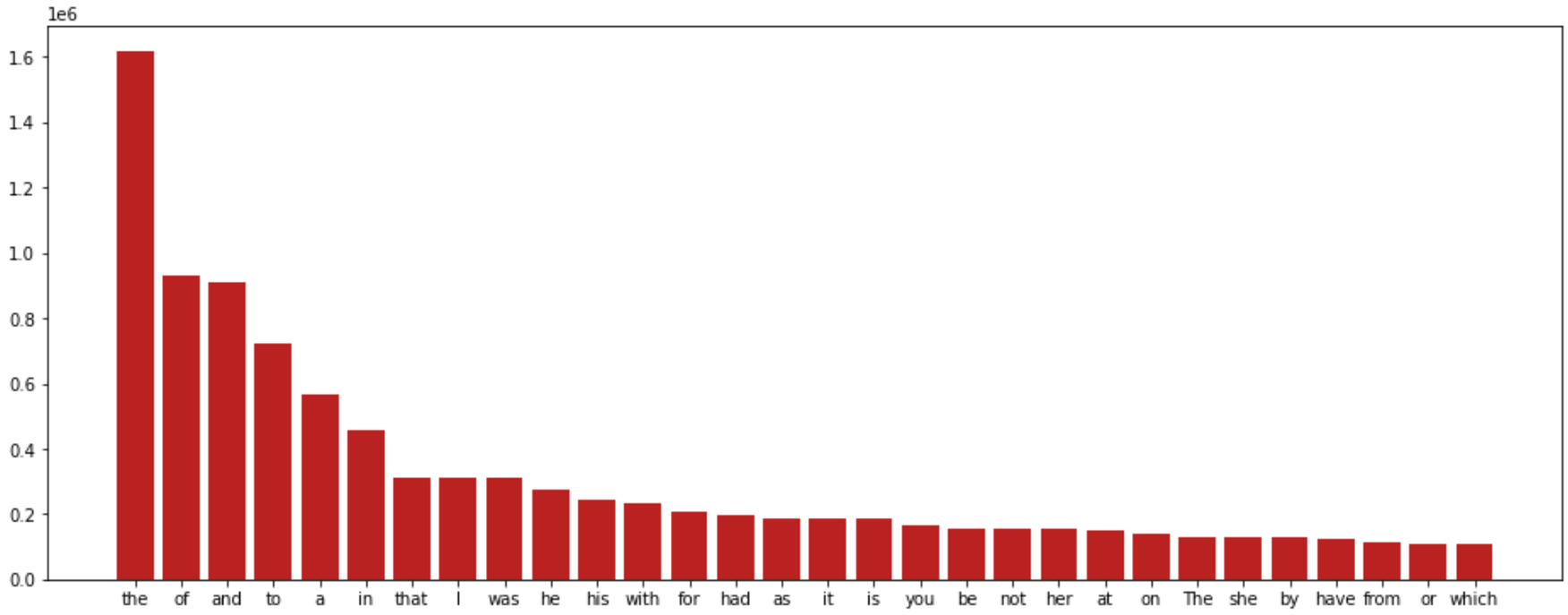
2.4. Plot the results

In [1]:
```
import matplotlib.pyplot as plt
import operator
print ('Plotting...')
freq = {}

# Read input and sort by frequency. Keep only top 30.
# Read Reducer Output
with open('./results/part-r-00000', 'rb') as csvfile:
    for line in csvfile.readlines():
        word, count = line.decode('UTF-8').split('\t')
        freq[word] = int(count)
srt = sorted(freq.items(), key=operator.itemgetter(1), reverse=True)[:30]

# Generate plot
plt.figure(figsize=(16,6))
plt.bar(range(len(srt)), [x[1] for x in srt], align='center', color='#ba2121')
plt.xticks(range(len(srt)), [x[0] for x in srt])
plt.show()
```

```
Plotting...
```



If everything is correct, the 3 most frequent words should be **`the`**, **`of`** and **`and`**.

3. Performance comparison

- We prepared a simple wordcount program in Python in this folder. Download it on your laptop (or the cluster NameNode) and test how long it takes to process these two datasets. **next exercise.**

  ```
  python wordcount.py < /pathtoyourfile/gutenberg_x0.1.txt
  python wordcount.py < /pathtoyourfile/gutenberg.txt
  ```

- For reference, it takes 5.724s for gutenberg_x0.1.txt in my local workstation and it takes 42.984s for gutenberg.txt.

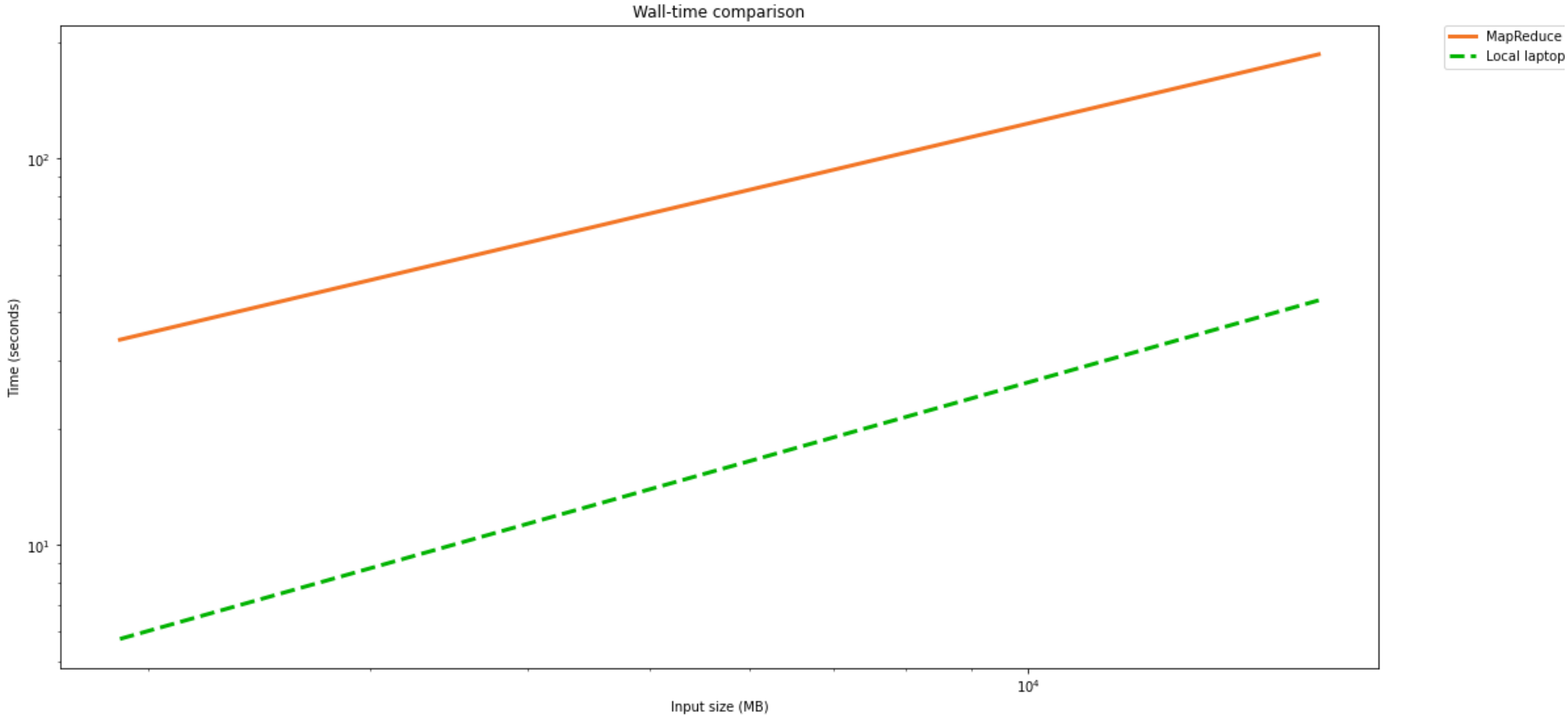
3.1 Plot

Compare the performance of the MapReduce vs the single-thread implementation of the word count algorithm for two different input sizes. Fill the time in seconds in the code below t

In [2]:
```
size_input = [1.9*10e2, 1.7*10e3] # the input size in MB
time_mapreduce = [34., 186] # replace 0s with the time (in seconds) for the corresponding inputs
time_locallaptop = [5.724, 42.984] # replace 0s with the time (in seconds) for the corresponding inputs


%matplotlib inline
# Import plot library
import matplotlib.pyplot as plt
# Plot
plt.figure(figsize=(18,9))
plt.plot(size_input, time_mapreduce, '#f37626', label='MapReduce', linewidth=3.0)
plt.plot(size_input, time_locallaptop, '#00b300', label='Local laptop', linewidth=3.0, linestyle='dashed')

plt.xscale('log')
plt.yscale('log')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.xlabel('Input size (MB)')
plt.ylabel('Time (seconds)')
plt.title('Wall-time comparison')
plt.show()
```



3.2. Discussion

We have run some more tests. Here we present the running time for 3 configurations on HDInsight, one workstation and one laptop. The figures below are indicative only, because the depends on several factors.

- **MapReduce v1**: no combiner with default configuration (1 reducer)
- **MapReduce v2**: no combiner with 8 reduce tasks
- **MapReduce v3**: using combiner with default configuration (1 reducer)
- **MapReduce v4**: using combiner with 8 reduce tasks
- **Workstation**: using a local workstation (server)
- **Laptop**: using a local laptop

4. Why MapReduce is not performing better than local computation for small inputs?
5. How can you optimize the MapReduce performance for this job?

Solution

We have run some more tests. Here we present the running time for 3 configurations on HDInsight, one workstation and one laptop. The figures below are indicative only, because the depends on several factors.

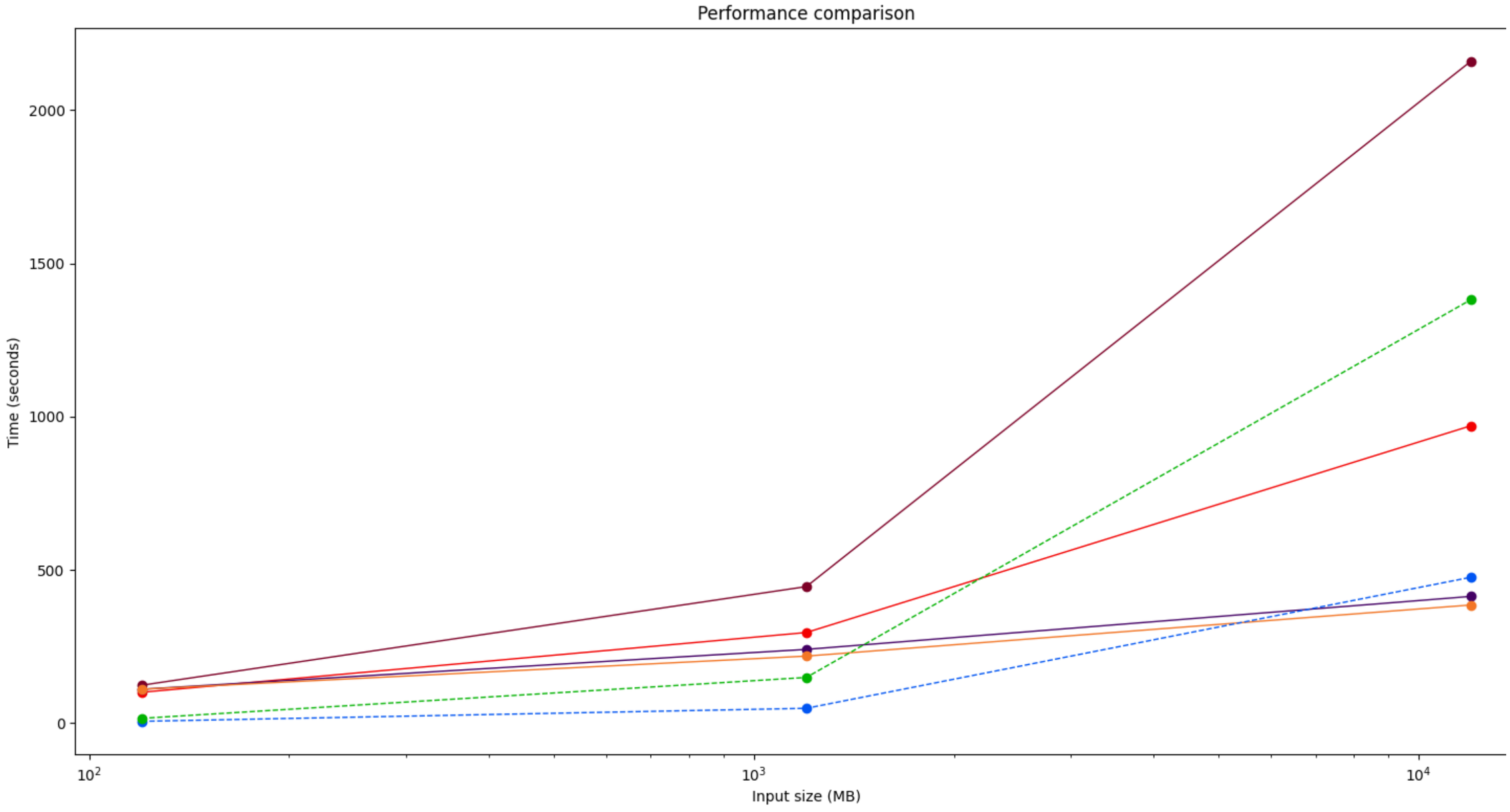
- **MapReduce v1**: no combiner with default configuration (1 reducer)
- **MapReduce v2**: no combiner with 8 reduce tasks
- **MapReduce v3**: using combiner with default configuration (1 reducer)
- **MapReduce v4**: using combiner with 8 reduce tasks

1. See figure below
2. For small inputs, local computation is faster
3. Using a combiner, MapReduce gets faster than a single workstation somewhere between 1GB and 10GB of data. This of course, really depends on the task and the configuration o
4. The shuffling phase adds a lot of overhead computation
5. A combiner drastically improves the performance for this job where the data to be shuffled can be reduced a lot

In [4]:
```
size_input = [1.2*1e2, 1.2*1e3, 1.2*1e4] # the input size in MB
time_mapreduce1 = [123, 445, 2160] # MapReduce v1
time_mapreduce2 = [100, 295, 970] # MapReduce v2
time_mapreduce3 = [110, 240, 413] # MapReduce v3
time_mapreduce4 = [110, 218, 385] # MapReduce v4
# 11.4, 127, 1332
#time_localstation = [11.4, 127, 1332] # our workstation (java)
time_localstation = [5.1, 47.7, 475.5] # our workstation
time_locallaptop = [14.9, 148.4, 1381.7] # our laptop


%matplotlib inline
# Import plot library
import matplotlib.pyplot as plt
# Plot
plt.figure(figsize=(18,9))
plt.plot(size_input, time_mapreduce1, '#7d0026', label='MapReduce v1', linewidth=1.0, marker='o')
plt.plot(size_input, time_mapreduce2, '#f30000', label='MapReduce v2', linewidth=1.0, marker='o')
plt.plot(size_input, time_mapreduce3, '#430064', label='MapReduce v3', linewidth=1.0, marker='o')
plt.plot(size_input, time_mapreduce4, '#f37626', label='MapReduce v4', linewidth=1.0, marker='o')

plt.plot(size_input, time_localstation, '#0055f3', label='Local workstation', linewidth=1.0, linestyle='dashed', marker='o')
plt.plot(size_input, time_locallaptop, '#00b300', label='Local laptop', linewidth=1.0, linestyle='dashed', marker='o')
```



In [5]:
```
size_input = [1.2*1e2, 1.2*1e3, 1.2*1e4] # the input size in MB
time_mapreduce1 = [123, 445, 2160] # MapReduce v1
time_mapreduce2 = [100, 295, 970] # MapReduce v2
time_mapreduce3 = [110, 240, 413] # MapReduce v3
time_mapreduce4 = [110, 218, 385] # MapReduce v4
# 11.4, 127, 1332
#time_localstation = [11.4, 127, 1332] # our workstation (java)
time_localstation = [5.1, 47.7, 475.5] # our workstation
```

```
plt.plot(size_input, time_mapreduce2, '#f30000', label='MapReduce v2', linewidth=3.0)
plt.plot(size_input, time_mapreduce3, '#430064', label='MapReduce v3', linewidth=3.0)
plt.plot(size_input, time_mapreduce4, '#f37626', label='MapReduce v4', linewidth=3.0)

plt.plot(size_input, time_localstation, '#0055f3', label='Local workstation', linewidth=3.0, linestyle='dashed')
plt.plot(size_input, time_locallaptop, '#00b300', label='Local laptop', linewidth=3.0, linestyle='dashed')

plt.xscale('log')
plt.yscale('log')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.xlabel('Input size (MB)')
plt.ylabel('Time (seconds)')
plt.title('Performance comparison')
plt.show()
```