Distributed systems Spring 2023

> Pramod Rao
>
> 2020111012

Directory structure

```
q2
├── inp.txt
├── mapper.py
├── reducer.py
└── runner_script.py
q3
├── inp.txt
├── mapper.py
├── reducer0.py
├── reducer.py
└── runner_script.py
```

Instructions

In order to run the runner_scripts, simply run `./runner_script.py` with appropriate execution permissions provided.
The arguments' usage info for the program can be found by running `./runner_script.py -h` or `./runner_script --help`

Here is a snippet from the output of the above help command:

```
positional arguments:
  jar         location to the hadoop streaming jar
  loc_in      location to the input file in the local FS
  hdfs_in     location to the input file to be put in HDFS
  hdfs_out    location to the output directory on HDFS
  hdfs_exec   location to the directory on HDFS containing the mapper(s) and
reducer(s)
```

Example usage:

`./runner_script.py /opt/hadoop-3.2.1/share/hadoop/tools/lib/hadoop-streaming-3.2.1.jar ./inp.txt input/ output/ ./`

Note that the directory arguments need to terminate with a `/` character

Questions

A typical program can be broken down into four main stages:

1. Pre-processing the input
2. Mapper stage with parallel mapper tasks
3. (Optional) Combine stage with parallel combines on the map output
4. Reduce stage with a single reduce task

Question 2 (Matrix multiplication)

Given matrices `A` and `B`, output the matrix `C` which is the result of the operation `C = A x B`.

Dimensions of `A`: `m x n`, Dimensions of `B`: `n x p`.

1. Pre-processing:

In this stage, we convert the input format in the following manner:

- Each element of the matrix `A` is mapped to the following comma-separated list: `0, p, n, i, j, A_ij`
- Each element of the matrix `B` is mapped to the following comma-separated list: `1, m, n, i, j, B_ij`

The following is the significance of each entry:

- ID of the matrix
- Corresponding dimension of the final matrix
- Inner product dimension
- Row index
- Column index
- Value

2. Mapper

The mapper reads each line in the above format, parses the integers and then performs the following mapping:

- Element `A_ij` is mapped to the key-value pair `(i,k) -> (0, j, n, A_ij)` for all `k: [0, p - 1]`
- Element `B_ij` is mapped to the key-value pair `(k,j) -> (1, i, n, b_ij)` for all `k: [0, m - 1]`

3. Combiner
Absent

4. Reducer

At this stage of the execution, each key `(i, j)` corresponds to a row of `A` and a column of `B`. We take their inner
product and obtain the final matrix element-wise.

**Note: Owing to the constraints of the question, we have assumed a single reduce stage suffices and so, have output the
matrix accordingly. If this was not the case, then we would require another round of reduce to collect the outputs per
row and output them in order.**

Question 3 (Monte Carlo)

Estimate the value of Euler's number `e = 2.71828`.

1. Pre-processing:

Given input as the number of simulations to be performed, we divide the simulations among the map tasks. A challenge
that comes with this the generation of the pseudo-random numbers. If we seed the mappers with the current time (which is

a commonly adopted technique), there is a chance that two different map tasks end up with the same seed leading to a
biased experiment. So instead, we write the number of simulations to be performed by the mapper and a random floating
seed as part of the input.

To summarize, the total required simulation count `N` is divided into multiple smaller simulation counts and supplied to
different map input files along with a random floating point seed.

2. Mapper

The mapper reads the number of simulations to be performed and the random seed supplied. It seeds the generated with the
given seed and then performs the experiment required number of times.

For each experiment, if the outcome is `n`, then, the mapper outputs a key-value pair of the form `(n, 1)`.

3. Combiner

Notice that directly performing the reduce stage (a single reduce task) would mean that we need to go through `N`
entries, where `N` is the total number of simulations. Instead, we perform an intermediate combine step which is very
similar in function to the reducer but performed locally on a map output.

Doing so, produced a speedup (`65s -> 53s` for simulation count of the order `1e7`)

4. Reducer

At this stage of the execution, the value is a frequency count of key. We perform a simple addition as part of the
reduce stage.

**Note: Owing to the constraints of the question, we have assumed a single reduce stage suffices and so, have output the
final result accordingly. If this was not the case, then we would need another round of reduce to perform an aggregate
on the previously considered final outputs**

Analysis:

Simulation count	Estimate of e	Percent error (taking ground truth e = 2.71828)	Time taken (s)
10	2.6	4.3513%	17.33
100	2.78	2.2705%	17.28
1000	2.743	0.9093%	17.48
10000	2.7176	0.0251%	17.48
100000	2.7188	0.0191%	16.41
1000000	2.7179	0.0140%	20.533
10000000	2.7185	0.0080%	42
100000000	2.718214	0.0025%	158

At `N = 1e7` the program switches to a distributed mode of computation and hence, would require a different analysis as compared to the preceding rows.
We see that the program has a good scaling factor given that a 10 times increase in simulation count from 1e7 to 1e8 only resulted in ~4 times the run-time.

As for the accuracy, we see that with 1e8 simulations, we get a percent error as low as `0.0025%` from the ground truth.