Timothy Chu
Michael Wong
CPE 369-01

Lab 7 Report

Sequential Implementation

Pseudocode:

```
For each JSON in the file, do
        Get the (x, y) location
        Store in HashMap associating the location to number of occurrences
End for

For each element in HashMap, do
        Write HashMap key and value
End for
```

Parallel Implementation

Map Pseudocode:

```
Key = each x, y pair
Value = 1;
Context.write(Key, Value)
```

Reduce Pseudocode:

```
Sum up the number of occurrences
Context.write(Key, Sum)
```

Analysis

1. Capture the overall performance of both implementations as a function of the size of input (in bytes, in number of JSON objects).

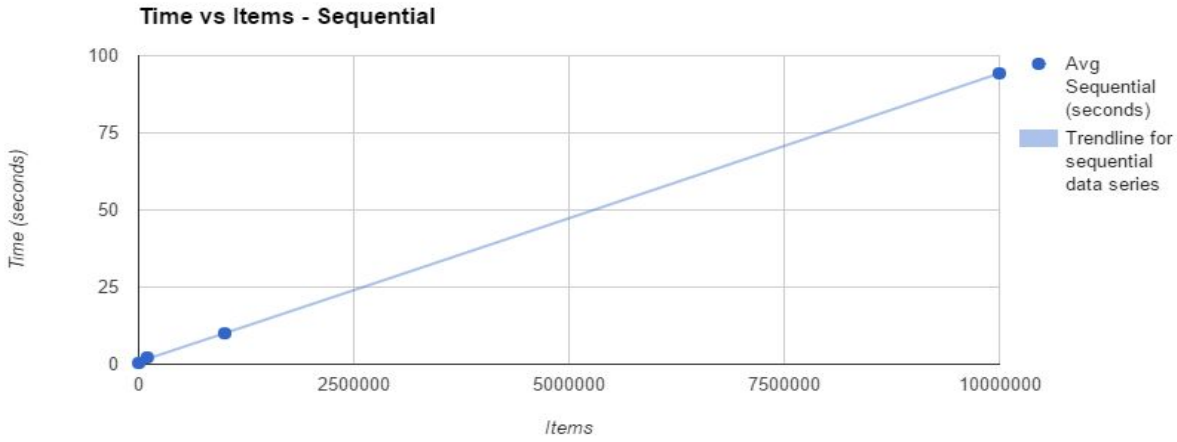
Overall performance of both implementations can be found in the chart below:

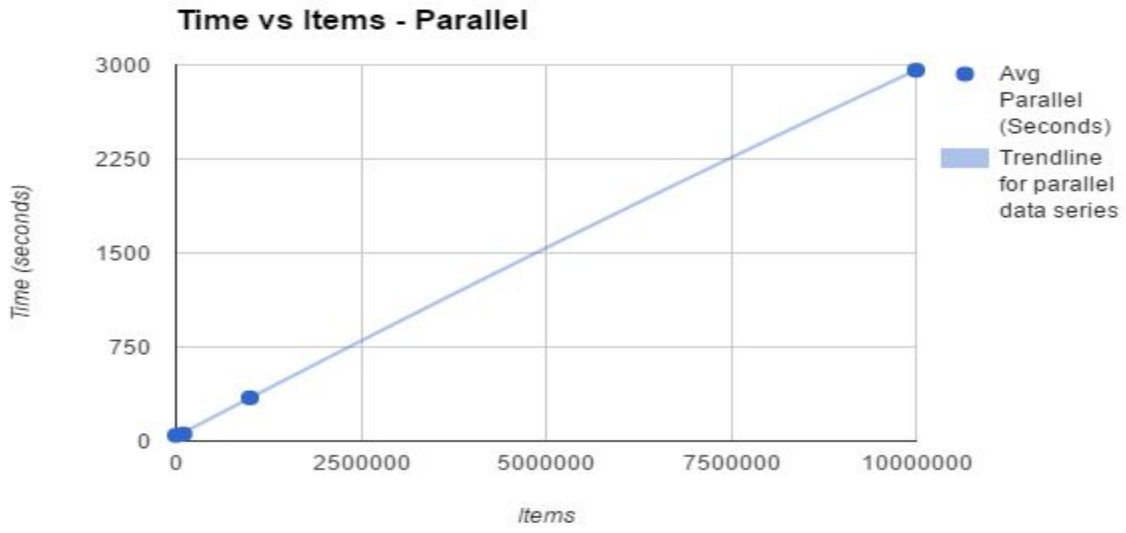
Items	Avg Sequential (seconds)	Avg Parallel (Seconds)
1000	0.4	45.137
100000	2.196	56.219
1000000	10.021	344.608
10000000	94.287	2958.007

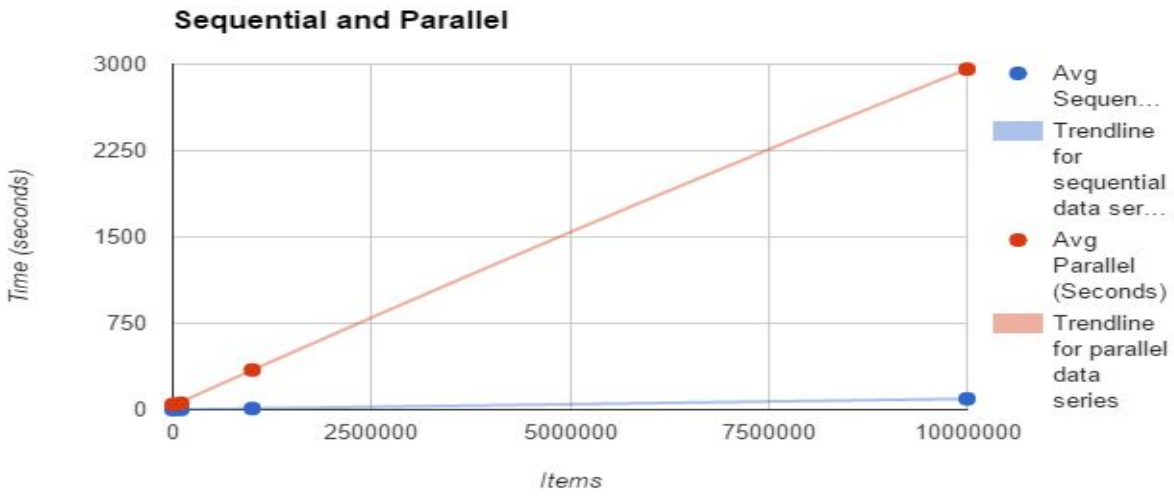
Overall, sequential ran faster than parallel in all scenarios.

2. Determine the performance trends for each of the two implementations (you can use your intuition here, or use regression).

The performance trends for each of the two implementations can be found in the following graphs:







3. Determine if there is an ”inflection point” - the size of input at which it is more advantageous to use MapReduce for this task, than it is to use a straightforward implementation.

There does not seem to be an inflection point with the sample size that we tested. Running 10 million objects on hadoop took 50 minutes, compared to the 95 seconds in sequential. A much bigger data size is needed to clearly see the inflection point. We predict that it will be more advantageous to use MapReduce when the data size grows too big to be stored on one machine.

The margin of error may be large due to the number of users on the Hadoop cluster as well, resulting in skewed results.