CSC462 Lab 3

Release

Blake Smith

May 30, 2020

1 CSC Lab 3: Analysis of Map-Reduce Performance

1.1 Sequential Map Reduce, A Baseline

Before proceeding with testing my implementation from Lab 2 I will first gather some metrics from the sequential implementation given in *mrseqential.go*. This baseline will be used to determine at what point the multi-process implementation becomes worthwhile and what the trade offs are in regards to space & time complexity.

1.1.1 The Phases of Map Reduce (In the Sequential Case)

In order to estimate the performance of a sequential map-reduce I will be collecting the following information at each step in the execution.

1. Read input files and pass into the map function, producing a collection of intermediate values.
    - Time taken to read in the input files and produce the intermediate collection.
    - Space required to store the intermediate values
    - Time taken to sort the intermediate values
2. Run Reduce on each key and create a single output file
    - Time taken to complete all reduce jobs and produce the full output

1.1.2 Collecting Stats for *mrsequential.go*

First I created a struct to keep all of the statistics.

```
type Stats struct {
        // total execution time
        TotalTime time.Duration
        // time to produce intermedates
        MapSeqenceTime time.Duration
        // (in-memory) space required to store intermediates (in bytes)
        IntermediateSpace int
        // time to sort the intermediates
        SortTime time.Duration
        // runtime of reduce jobs, including storage of the results
        ReduceSequenceTime time.Duration
        // number of bytes from all the keys (including duplicates)
        NumKeyBytes int
        // number of bytes from all the values
        NumValBytes int
        // number of KV pairs processed
        NumRecords int
        // number of keys after grouping (number of calls to reduce
        NumKeys int
}
```

After collecting the data I simply output the *JSON* encoding of the struct to *stdout*.

```
{
        "TotalTime":703120545,
        "MapSeqenceTime":371379906,
        "IntermediateSpace":3150705,
        "SortTime":194342963,
        "ReduceSequenceTime":127343518,
        "NumKeyBytes":2526757,
        "NumValBytes":623948,
        "NumRecords":623948,
        "NumKeys":22107
}
```

Script to Run Various Senarios

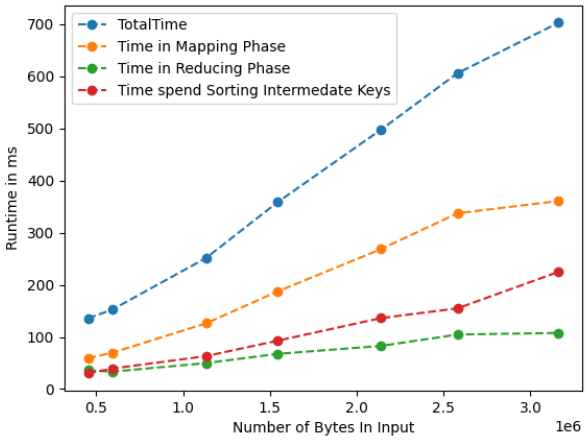
Next I wrote a *Python* script to run various scenarios and serialize the results to be analyzed later. This script will run mrseqential.go on different numbers of input files and store the results for each case in a *.json* file

```
import os
import json
from pathlib import Path

textfiles = list(Path(__file__).parent.glob('pg-*'))
of = open('sequential_stats.json', 'a')

for i, txt in enumerate(textfiles[:-1]):
    running = textfiles[:i+1]
    print(f"running mrsequential.go with {len(running)} files")
    cmd = "go run ./mrsequential.go wc.so " + " ".join([str(p) for p in
running])
    stream = os.popen(cmd)
    output = stream.read()
    _json = json.loads(output)
    _json['label'] = f'seq_{len(running)}'
    jsonstr = json.dumps(_json, indent=4, sort_keys=True)
    of.write(jsonstr + '\n')
```

From the following figure you can see that the runtime scales linearly with the input size as expected.



1.2 Distributed Map Reduce