Multicultural City
(A Parallel Approach)

Cluster and Cloud Computing
(COMP90024)

Nithin Mathew
1328669
Ritwik Giri
1301272

April 4, 2022

1 Problem Statement

To implement a parallelized application, which leverages the power of the University of Melbourne's HPC (High-Performance Computing), SPARTAN. The application will count the number of different languages used in the tweets in the given location grid/mesh for Sydney using the Twitter dataset and calculate the multicultural nature present in the city.

2 Data Source

1. **bigTwitter.json** - Twitter data set of approximately 40 million tweets containing the location and language information.
2. **sydGrid.json** - JSON of Latitudes and Longitudes of the targeted areas of Sydney. It is represented in the grid of 4*4 matrix.

3 Key Assumptions

The following assumptions are made as per the provided specification document.

1. Tweets with null or undefined (und) language attributes are ignored
2. Tweets with no location information are ignored
3. If a tweet occurs in the border of the two cells, then it is assumed that it occurred in the cell to the left or the cell below
4. If a tweet occurs on the leftmost or upper border of a particular cell, then it has occurred in the same cell

4 Tools used

1. **Python3.9** - With its simplified syntax and ease of usage, python is used in the implementation.
2. **Mpi4py** - This library enables the python program to utilize multiple computing resources. It allows the python applications to leverage Message Passing Interface (MPI) to exploit processors on clusters or supercomputers.
3. **SLURM** - It allocates nodes to user, to perform their jobs. It acts as a resource manager for the processors.

5 Implementation Approach

5.1 Resource Allocation

The SPARTAN is accessed from the Linux based head node using the resource management scheduler SLURM. The python package is implemented once for each of the following resources - **1 node 1 core , 1 node 8 core, 2 node 8 core (4 cores per node)** .

5.2 Invocation

The SLURM files for the respective resource allocations are executed from the user's head node, which invokes the python package **app.py** to analyse region-wise tweets (Refer Figure 1 for process flow).
The python package executes the **twitterRegionAnalysis.py**, which performs parallel implementation of language-region analysis. It distributes the tasks between master (rank = 0) and child processors and is responsible to combine the processed language region analysis data from the child and master nodes.(Refer Figure 1)

5.3 Master - Child Configuration

1. The processor with rank = 0 is assigned as the Master processor. Inside the master, the bigTwitter.json file is split into segments. Each segment is then processed by the child processors.

2. Analysis output from each child is passed to master, which aggregates the different outputs and generate the final merged output.

5.4 Implemented Functions

- **processGridObject** - Convert the location JSON file into a list of dictionaries

- **getTweetRegion** - Cater the boundary conditions requirements as mentioned in the assumptions

- **getLang** - Add the location information to the output region analysis dictionary

- **processTweets** - Process the bigTwitter.json iteratively. Each processor reads the part of the file as per its associated segment received from the master.

- **mergeResults** - Combine the outputs from each child and master nodes and send the results back to the head node.

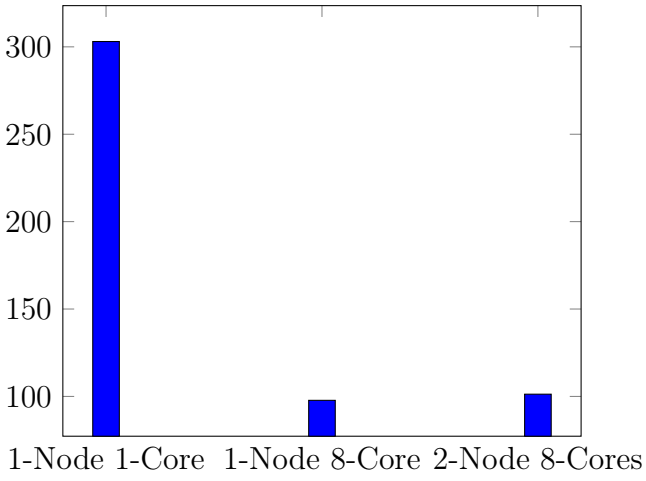
6 Output

Using the MPI framework we were able to parallelize the workload and achieve the results in lesser time by distributing the task across multiple cores. To test the extent of the improvement we ran the workload on three different resource configuration and used time taken to run the task as a metric for evaluation.

1. Baseline (1 node 1 core): The workload was run in a serialised manner and we obtained the results in **303 seconds**.

2. Test 2 (1 node 8 cores): Task was distributed amongst 1 master processor and 7 child processors to achieve parallelisation. This reduced the time taken **97 seconds** (3x faster than the baseline).

3. Test 3 (2 nodes 8 cores- 4 core per node): Similar to the earlier case we distributed the task among 1 master processor and 7 child processors to achieve parallelisation. The time observed was comparable to the Test 2 results. Time taken was **101 seconds** (3x faster than the baseline).



Time taken with different resources

The detailed information about the results from each test runs can be found in the attached project (.zip) under **FinalOutputs**.

7 Observations and Challenges

7.1 Challenges

1. Memory: Since the json file to be processed was too large to be read into memory, it had to be broken down into smaller chunks that could be read iteratively.

7.2 Observations

1. **Amdahl's Law** - 8 core is not 8x faster than the single core. Implementation is a combination of both sequential and parallel computing. Even the use of MPI and n extra cores does not make the compute n times faster. Refer the fig 1 for the process flow.

2. **Network Latency** might play a role when the number of nodes are increased. But this does not play a major role in our results, where 1 node 8 core is slightly faster,by 4 seconds, than 2 nodes 8 cores implementation. This is likely due to the fact that the data-flow from child processes to master process occur only once in the end of the process. In use-cases like simulation where there is a constant communication between the processes the network latency plays a more significant role.

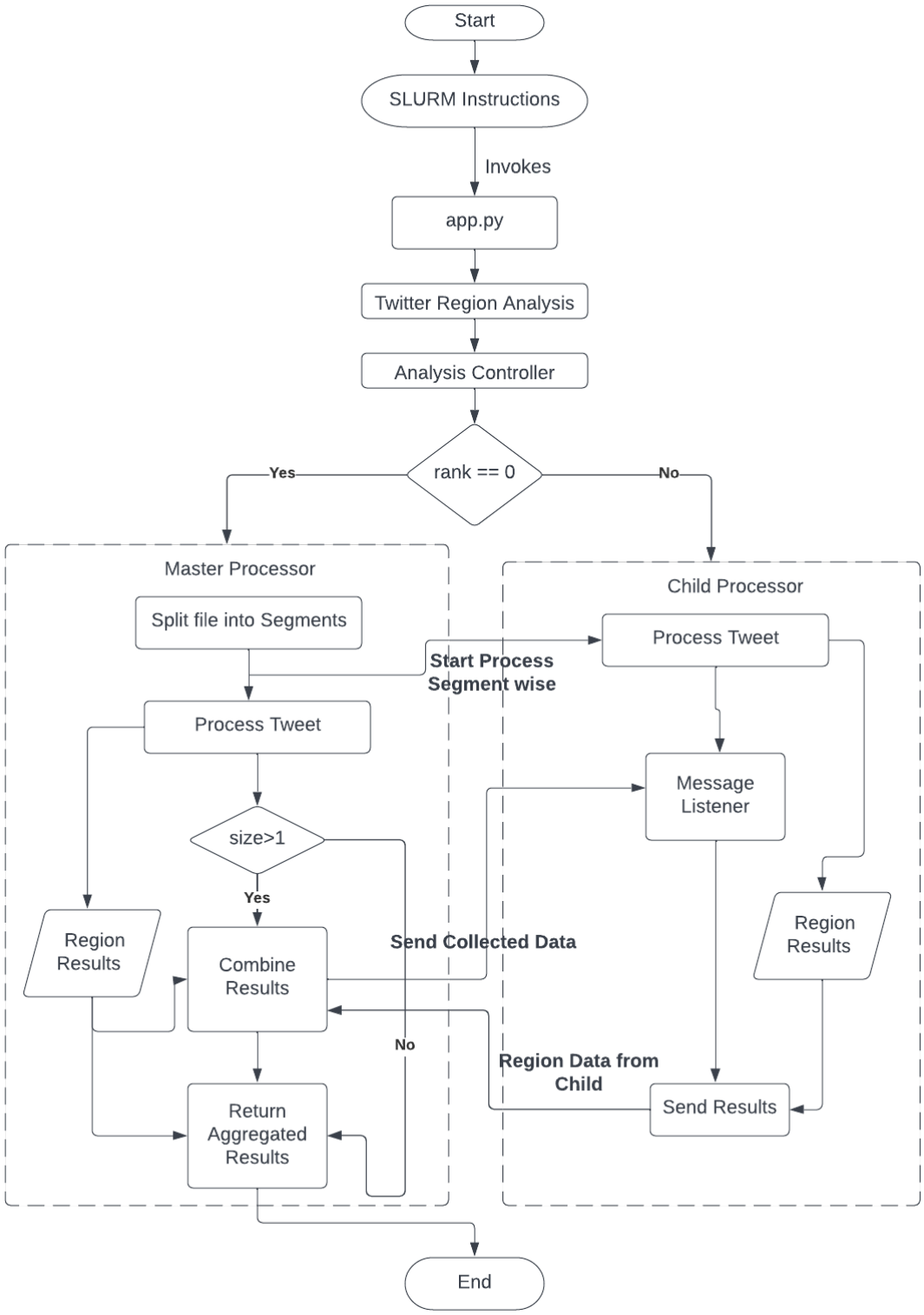


Figure 1: Process Flow Diagram