MapReduce Implementation of Page Rank

COL730 Assignment 3

Dakshit Babbar | 2020EE30163

November 2022

1 Objective

There is a huge number of web-pages on the internet and search engines like Google, are expected to list out some of these pages that are related to some keywords typed in the search bar. But these pages shouldn't be listed in a haphazard manner, there should be a proper page ranking according to which these would be listed out. The rank of a page depends on its importance, and its importance is measured differently in different algorithms.Google's page ranking algorithm uses the number of pages that link to a given page, as the measure of its importance.

We describe the algorithm in brief here. All page ranks are maintained in a vector I and are initialised to be $1/n$ where n is the total number of pages. The following equation is run for 50-100 iterations,

$$I^{k+1} = \alpha H I^k + \alpha A I^k + (1-\alpha)\frac{1}{n} 1 I^k$$

Where H is the matrix with each row as describing the inlink-contribution to the corresponding page, A is the matrix with all elements zero except for the columns corresponding to those pages which do not have any outlinks, in which case the elements are $1/n$ and 1 is the matrix with all 1's.

In this assignment we are expected to implement Google's page rank algorithm using the famous MapReduce programming paradigm. The following are the three main objectives of this assignment,

- Implement pagerank using the given C++ MapReduce Library.
- Implement our own mapreduce library using MPI and use this to implement pagerank.
- Implement pagerank using the given MPI MapReduce Library.

We have also been given some pagerank outputs, we are expected to check the correctness of our pagerank implementation using these outputs.

2 Using C++ Library

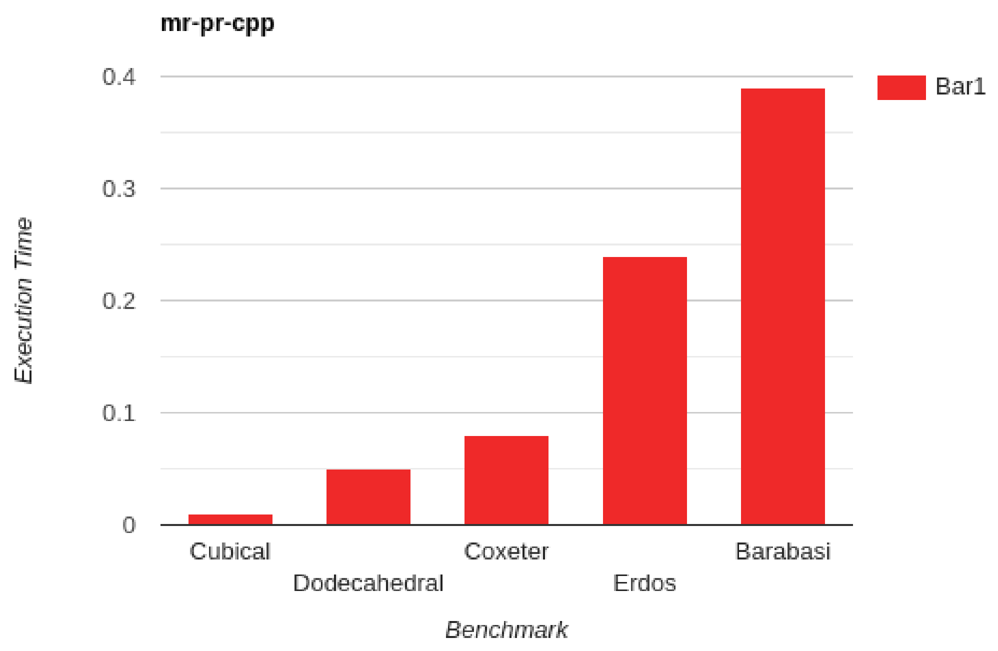
Here we were provided with a C++ MapReduce library and we had to implement the pagerank algorithm using this library. Here we were to define three classes, DATA_SOURCE to distribute data, MAP_TASK to perform mapping operation, REDUCE_TASK to perform reduction operation.

The following is the pseudo-code of the algorithm used:

```
//read input file
//make adjacency list out of it

int num_iter = 50;
for(int i=0; i<num_iter; i++){
    DATA_SOURCE
    //this function splits the adjacency list into chunks
    //and distributes to mappers

    MAP_TASK
    //this function maps the adjacency list to key value pair of the form:
    //(key, value)
    //where key is the page number and value is the contribution
    //from one of its inlinks.

    REDUCE_TASK
    //this function takes all the values of a single key
    //adds them up to get the final updated weight
}
```

We used the above approach to calculate the pageranks of the following four datasets and compared the respective execution times.

Dataset	**Execution Time(s)**
Cubical	0.01
Dodecahedral	0.05
Coxeter	0.08
Erdos10000	0.24
Barabasi20000	0.39



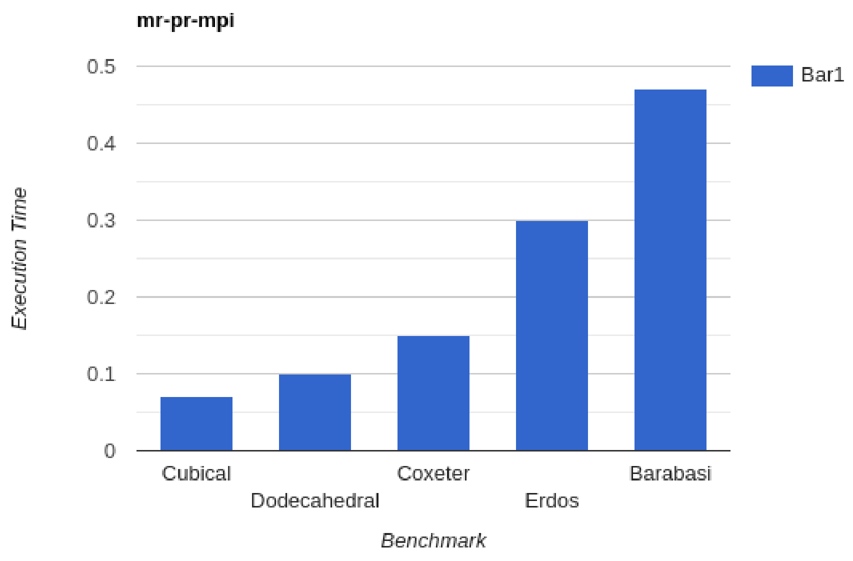
3 Developing and Using MPI Library

Here we had to develop our own MapReduce Library using the MPI programing paradigm and then use this library to implement the pagerank algorithm. The following is the pseudocode of algorithm used.

```
MPI_Init();
if(rank==0){
    //read input file
    //make adjacency list
    //output this into an auxiliary file
}

int num_iter = 50;
for(int i=0; i<num_iter; i++){
    MR_datasource(...)
    //distribute the data in the auxiliary file to all the processes
    //all chunks in form of strings

    MR_map(...)
    //convert the input string into vector
    //emit key value pairs of the form
    //(key, value)
    //where key is the page number and value is the contribution
    //from one of its inlinks.

    MR_collate(...)
    //collect all the key value pairs into process 0
    //redistribute to all the reducers inform of strings

    MR_reduce(...)
    //perform reduction operation
    //sums up all the values of the same key/page

    MR_gather(..)
    //gather final output from all processes
    //updates the ranks vector
}

MPI_Finalize();
```

We used the above approach to calculate the pageranks of the following four datasets and compared the respective execution times.

Dataset	**Execution Time(s)**
Cubical	0.07
Dodecahedral	0.10
Coxeter	0.15
Erdos10000	0.33
Barabasi20000	0.47



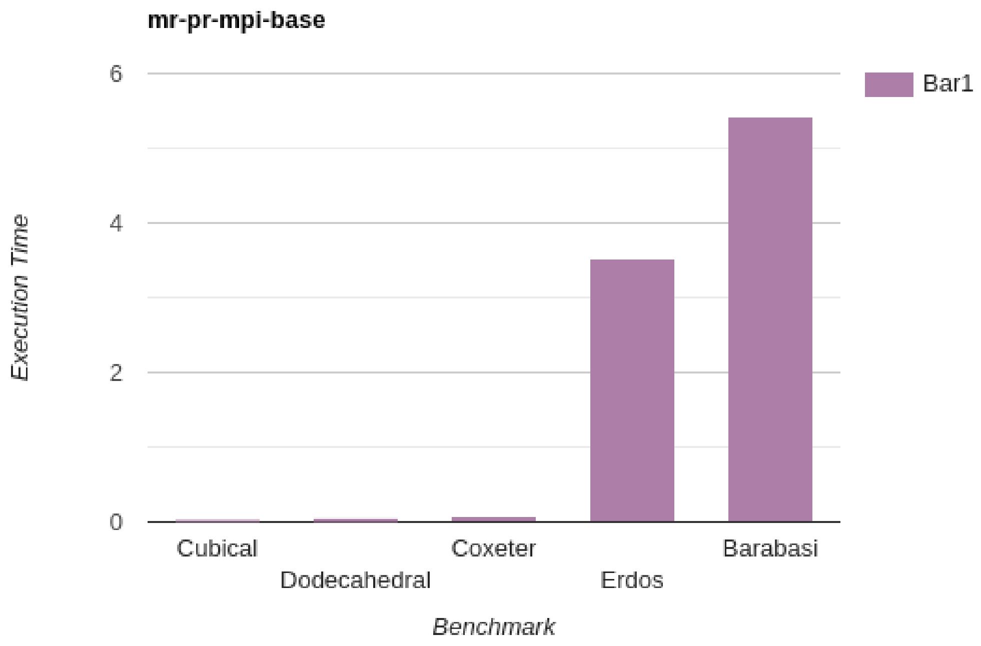
4 Using MPI Library

Here we were given an MPI MapReduce library and were asked to implement the pagerank algorithm using it. The following is the pseudo-code of the algorithm used.

```
MPI_Init();
if(rank==0){
    //read input file
    //make adjacency list
    //output this into an auxiliary file
}
int num_iter = 50;
for(int i=0; i<num_iter; i++){
    //distribute the data in the auxiliary file to all the processes
    //perform mapping operation
    mr->map(...)

    //collect all the key value pairs and shuffle
    //redistribute to all the reducers
    mr->collate(...)

    //perform reduction operation
    mr->reduce(...)

    //gather final output from all processes
    mr->gather(...)

    //update the ranks
    mr->scan
}
MPI_Finalize();
```

We used the above approach to calculate the pageranks of the following four datasets and compared the respective execution times.

Dataset	**Execution Time(s)**
Cubical	0.03
Dodecahedral	0.05
Coxeter	0.07
Erdos10000	4.12
Barabasi20000	7.53



5 Conclusions

The following observations can be made from the above plots.

- In TASK1 we have used a C++ library which eventually uses pthreads to launch different processes on the same machine. Hence, there is shared memory among the processes in TASK1. Due to this there is no communication overhead. Hence we get faster execution times in TASK1 as compared to those in TASK2 and TASK3
- In TASK2 we have implemented our own MapReduce library with only those functions which are to be used in the pagerank implementation. Due to this we get faster execution times in TASK2 as compared to those in TASK3. But, because we are using MPI here, there is no shared memory structure and hence there is an extra communication overhead.
- In TASK3 we have used an already implemented MPI library for mapreduce hence there is an extra communication overhead along with some extra functions which will be called during execution and hence we get the maximum execution times in this task.