# PDC SEMESTER PROJECT

Submitted to : Sir Adil Rehman
Section CS-Z

BY:
Muhammad Tallal Eatazaz 21i-0637
Shahzaib Irfan 21i-0841
Aimen Safdar 21i-0588

# Table of Contents

# Challenges faced:

- ## Challenges faced during Preprocessing:

  During the pre-processing phase of our project, we encountered several challenges related to handling node representations and data conversions. One significant challenge involved working with string-based node identifiers while needing to process them as integer values within our existing codebase. This required us to carefully manage and synchronize the string and integer representations of each node throughout the data processing pipeline.

  Another challenge arose when we needed to input start and end nodes as string identifiers but subsequently convert them into their corresponding integer values for use in the backend algorithm. This conversion process was critical for seamlessly integrating our input data with the computational logic of the algorithm.

  After processing, when presenting the results, we faced the task of converting the computed integer node values back into their original string identifiers to maintain consistency and readability in the output.

  Furthermore, a technical hurdle emerged when we needed to broadcast the integer-string value pairs of each node using MPI, especially considering that MPI does not inherently support broadcasting unordered maps. To address this, we implemented a strategy to serialize and deserialize the data into a single array, facilitating efficient data transmission and synchronization across processes after the root process reads the data from the file.

  These challenges required meticulous handling of data representations and efficient serialization techniques to ensure compatibility and smooth operation within our distributed computing environment. Through innovative solutions and careful data management, we successfully overcome these obstacles to achieve accurate and reliable data processing and analysis.

- ## Challenges faced during Implementation:

We were initially assigned the implementation of the k-shortest algorithm in a serial manner, which proved to be relatively straightforward compared to the subsequent parallelization phase. Our task then involved parallelizing the k-shortest algorithm using OpenMP and MPI following the successful completion of the serial version.

During the parallelization process using OpenMP, we encountered challenges related to optimizing the code for efficient parallel execution. Ensuring that the parallelized code yielded tangible performance improvements without introducing excessive overhead due to thread creation was particularly demanding and required careful optimization.

Additionally, while working with MPI, converting nested vector pair data structures and efficiently broadcasting and reconstructing this data posed significant challenges. Managing the complexity of data transmission and reconstruction across distributed processes added an additional layer of complexity to the implementation.

These challenges emphasized the importance of meticulous design and optimization strategies to achieve effective parallelization using OpenMP and MPI. Addressing these hurdles required thoughtful consideration of algorithmic efficiency and data management techniques to maximize performance and scalability in a distributed computing environment.

- ## Challenges faced during testing:

The testing phase posed significant challenges primarily due to the extensive volume of datasets involved. The process demanded a substantial investment of time and patience as the program

meticulously computed the K-shortest paths. The considerable computational workload required to derive these paths called for a systematic and patient approach to ensure accurate and comprehensive results. This phase highlighted the importance of resource management and efficient algorithmic design to navigate through the complexity of the task at hand.

# Optimizations

We shall be comparing two implementations of the k-shortest path algorithm: a serial version and a parallel version, and highlighting the optimization strategies employed in the latter to enhance performance and scalability.

In the serial version, the algorithm follows a sequential execution model, processing each node and its neighbors one at a time. Dijkstra's algorithm is implemented using a priority queue to explore paths based on accumulated weights, giving more importance to shorter paths discovered along the way. This version performs operations in a single-threaded manner, including file reading and path exploration, without using parallelism.

On the other hand, the parallel version introduces several optimization tactics to enhance computational efficiency. By integrating OpenMP directives,  it achieves intra-node parallelism within Dijkstra's algorithm loop. This approach enables multiple threads to concurrently explore adjacent nodes, effectively utilizing multiple CPU cores to accelerate path exploration.

Furthermore, it also utilizes MPI  to enable cluster computing across multiple processes, supporting inter-node communication and parallel execution on the cluster setup.

One of the key optimization strategies in the parallel version is hybrid parallelization, which combines shared-memory (openMP) and distributed-memory (MPI) parallelism. This hybrid approach utilizes local multicore CPUs  within each node and coordinates parallel execution across distributed nodes  to efficiently process large datasets. Moreover, it focuses on optimizing compute-intensive tasks, such as path exploration, while

maintaining sequential I/O operations for file reading and data initialization, which are less amenable to parallelization.

By introducing concurrency via parallelism and using distributed computing techniques, the parallel version maximizes resource utilization and scalability, making it well-suited for large-scale graph traversal tasks in parallel and distributed computing environments. These optimization strategies collectively contribute to improved performance and efficiency compared to the purely sequential approach of serial version, demonstrating the benefits of modern parallel computing techniques in accelerating graph-based algorithms like the k-shortest path algorithm.

# Results

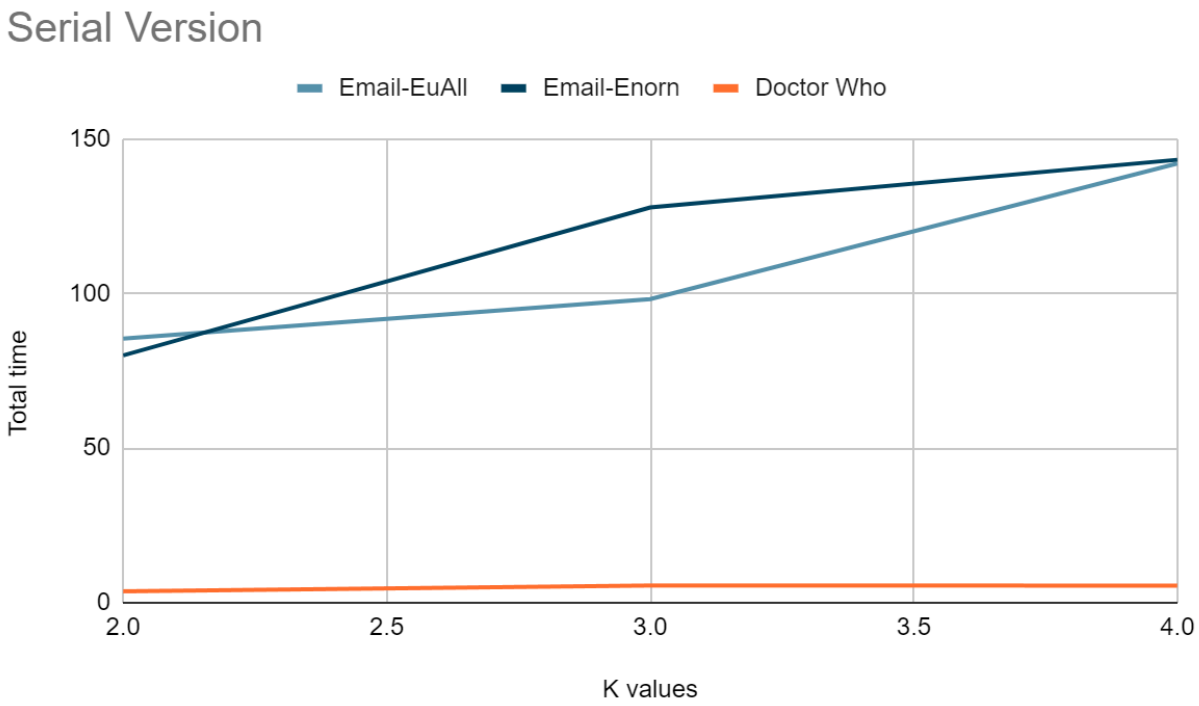
## 1. Execution Time:

- Serial Execution:

  The results below are shown in seconds.

| | Doctor Who | | Email-Enron | | Email-EuAll | |
|---|---|---|---|---|---|---|
| K | Total | Avg | Total | Avg | Total | Avg |
| 2 | 3.67932 | 0.367932 | 79.97841 | 7.99784 | 85.42488 | 8.542488 |
| 3 | 5.55513 | 0.555513 | 127.85282 | 12.785282 | 98.19901 | 9.819901 |
| 4 | 5.52336 | 0.552336 | 143.25916 | 14.325916 | 142.03820 | 14.203820 |

  The following graph shows the comparison between the total time taken with respect to the value of K. As the value of K increases, so does the total time taken for the serial version of K-shortest Algorithm.



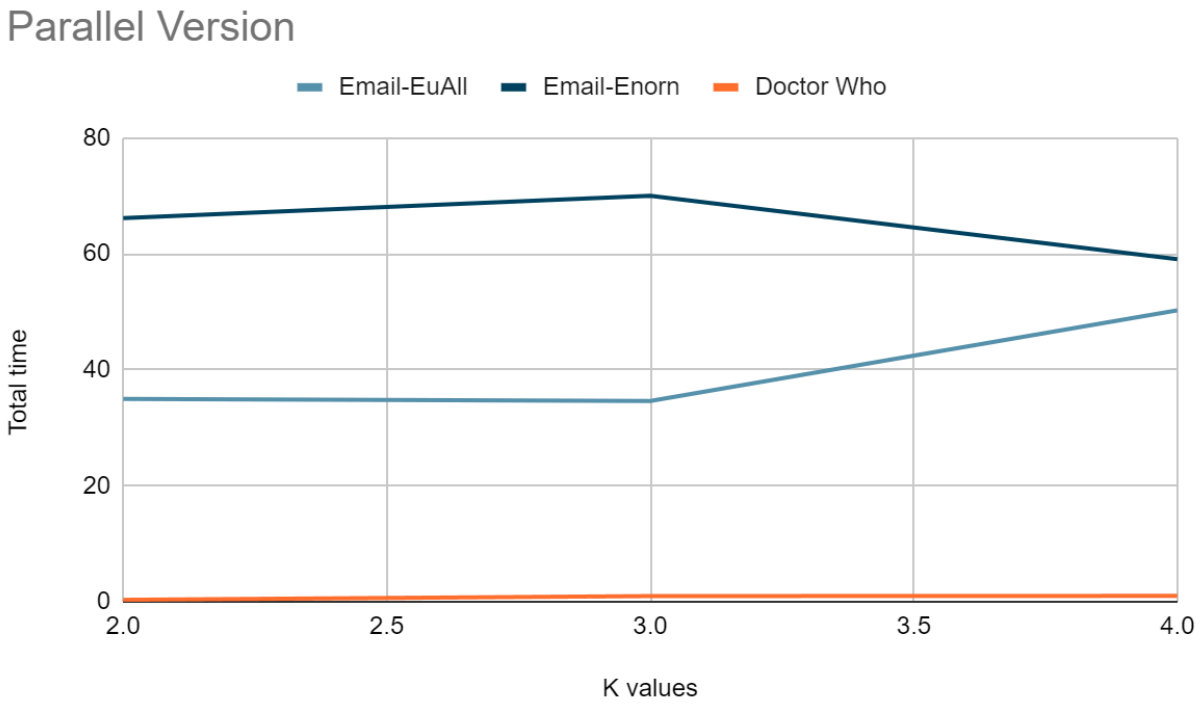

- Parallel Execution (n = 10):

  The results below are shown in seconds.
  Compared to the serial version of the 3 datasets, we can see that through parallelization the time for k - shortest algorithm has been reduced. The speed up can be observed in the next category.

| | Doctor Who | | Email-Enron | | Email-EuAll | |
|---|---|---|---|---|---|---|
| K | Total | Avg | Total | Avg | Total | Avg |
| 2 | 0.31975 | 0.031975 | 66.15575 | 6.615575 | 34.99660 | 3.499660 |
| 3 | 0.96763 | 0.096763 | 70.01154 | 7.001154 | 34.60641 | 3.460641 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 1.00686 | 0.100686 | 59.09210 | 5.909210 | 50.2565 | 5.02565 |

The following graph compares the total execution time with respect to the value of K for the parallel version of K-shortest Algorithm. Compared to the graph of the serial version, we can observe that the total execution time has decreased remarkably for the datasets but Email Enron still takes the longest time to achieve any results.




## 2.Speedup:

The speedup from the parallelization strategy can be found through the formula:

$$S(k) = T_{serial}(K) \ / \ T_{parallel}(K)$$

| K | Doctor Who | Email-Enron | Email-EuAl |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 2 | 11.506 | 1.20894 | 2.44095 |
| 3 | 5.7410 | 1.82346 | 2.83760 |
| 4 | 5.7081 | 2.42434 | 2.82626 |

Observing the table above we can deduce that parallelizing affected all datasets positively, i.e. we were able to see a considerable amount of speed up utilizing MPI and openMP.

## 3. Efficiency:

The efficiency will be calculated through the following formula:

$$E(k) = S(k) / p$$

*Where p = number of processes used in parallelization.*

| K | Doctor Who | Email-Enron | Email-EuAll |
|---|---|---|---|
| 2 | 1.1506 | 0.12 | 0.24 |
| 3 | 0.5742 | 0.18 | 0.28 |
| 4 | 0.5708 | 0.24 | 0.28 |

## 4. Scalability:

We will be analyzing the scalability utilizing graph density.

- Serial Version

| K | Doctor Who | Email-Enron | Email-EuAll |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 2 | 0.019. | 0.1368 | 0.4 |
| 3 | 0.0159 | 0.05 | 0.571 |
| 4 | 0.0211 | 0.3729 | 0.01617 |

- Parallel Version

| K | Doctor Who | Email-Enron | Email-EuAll |
|---|---|---|---|
| 2 | 0.0188 | 0.041 | 0.01617 |
| 3 | 0.0180 | 0.037 | 0.2222 |
| 4 | 0.0133 | 0.0455 | 0.1905 |

From these tables we can determine which graphs were densely populated and which were sparse; a denser graph means increased connectivity and greater complexity compared to that of a sparse graph. There isn't a pattern in any of these tables as these values all depend on the start and ending nodes which are being chosen randomly.