INF3203 - Distributed Systems
Assignment 1 - MapReduce and PageRank

Jens Christian Valen Leynse
jeley4465@uit.no
Tromsø, Norway
UiT

Bjørn Helge Kværnmo
bkv005@uit.no
Tromsø, Norway
UiT

March 2, 2025

1 Introduction

In this assignment, we will implement a simplified PageRank algorithm using the MapReduce programming model. The primary goal is to evaluate the scalability and performance of the algorithm with varying configurations of mappers and reducers. The implementation is built on a Python-based MapReduce framework that uses SSH and NFS for distributed communication between multiple nodes. We will also compare the results with the word count problem to understand the differences in computational complexity and scalability.

This report outlines the problem, explains the method to solve, the experimental setup, results, and analysis of the PageRank algorithm's performance under different configurations.

2 Problem Description

The **PageRank** algorithm ranks the importance of nodes in a graph based on their incoming links. Given a graph where **nodes** represent websites and **edges** represent hyperlinks, the simplified PageRank of a node u is calculated as:

$$\text{PageRank}(u) = \sum_{v \in B_u} \frac{\text{PageRank}(v)}{L(v)}$$

where:

- B_u is the set of nodes linking to u.
- $L(v)$ is the number of outgoing links from node v.

Initially, the PageRank of all nodes is set to 1. The algorithm iteratively updates PageRank values until convergence. However, for this assignment, we will focus on a **single iteration** of the algorithm.

3 Method

The implementation uses the MapReduce programming model to distribute the computation between multiple nodes. The framework divides the task into two phases:

Map Phase:

Processes each input data point, which consists of a node (representing a webpage) and its outgoing links.

For each node, the mapper calculates the rank contribution for every outgoing link and emits intermediate key-value pairs.

Reduce Phase:

Aggregates the intermediate results from the Map phase to compute the final PageRank values for each node.

The MapReduce framework uses SSH and NFS for communication and coordination between nodes. The implementation is tested on the ificluster infrastructure, which provides a distributed computing environment suitable for evaluating the scalability and performance of the PageRank algorithm.

4 Experiments and Results

The experiments evaluate the scalability of the PageRank algorithm with varying numbers of mappers and reducers, as well as different dataset sizes. (The mini dataset was excluded from the results due to inconsistencies and its lack of significant impact on the differences in the data output) The results are compared with the word count problem to highlight differences in computational complexity.

We conducted experiments using configurations with 2, 4, 8, and 16 mappers and reducers in a matrix-like analysis. This approach allowed us to explore everything from baseline performance to scaling issues and performance overheads associated with increasing the number of mappers and reducers.

4.1 Experiments

To determine which configuration results in the best execution times, we conducted three experiments. These experiments are sufficient for this assignment to analyze execution times variations when the number of mappers and reducers is either imbalanced or equal.

1. **Many reducers with few mappers**
2. **Many mappers with few reducers**
3. **Balanced/equal configurations**

Many reducers with few mappers: When the number of reducers is greater than the mappers, the reducers will stay idle while they are waiting for the mappers to complete their tasks. This leads to longer execution times and wastes resources.

Many mappers with few reducers: In this scenario, Too many mappers will place a heavy load on the limited number of reducers, creating a bottleneck. As a result, the overall execution slows down due to the reducers struggling to process large volumes of data.

Balanced/equal configurations: When mappers and reducers are equal it will generally yield the best performance. this setup minimizes the idle time and reduces the bottlenecks which leads to improved efficency and execution.

4.2 Results

The results are summarized in the following tables and graphs.

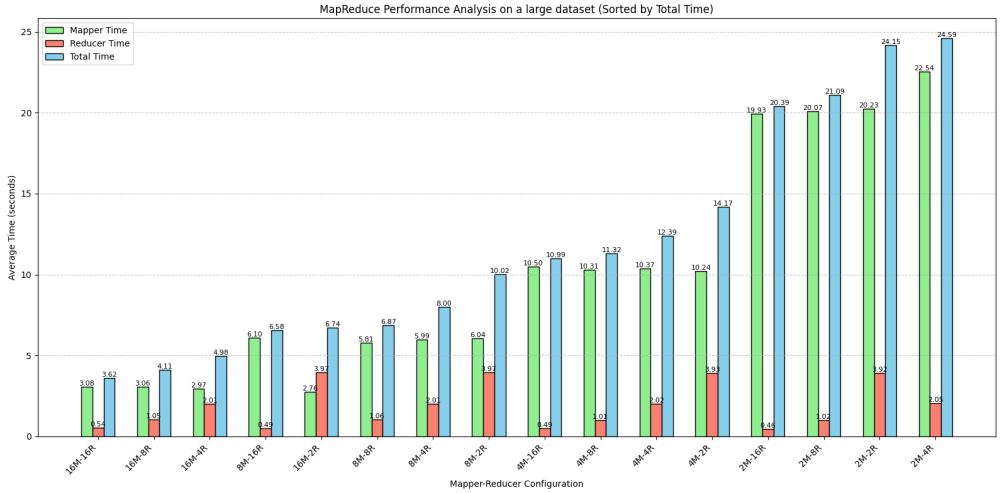

Figure 1: Scalability of PageRank large datasets with varying mappers and reducers.

The time complexity of PageRank scales with the number of mappers and reducers due to the parallel nature of the MapReduce framework. Observations include:

- **Increasing Mappers:** As the number of mappers increases, the workload is distributed between more nodes, reducing the overall computation time. For example, with 16 mappers and 16 reducers, the total execution time drops to 3.64 seconds, compared to 24.80 seconds with 2 mappers and 2 reducers.
- **Increasing Reducers:** Increasing reducers also improves performance, but the gains diminish as the number of reducers exceeds the number of mappers. For instance, with 16 mappers, increasing reducers from 2 to 16 reduces the total time from 6.89 to 3.64 seconds.

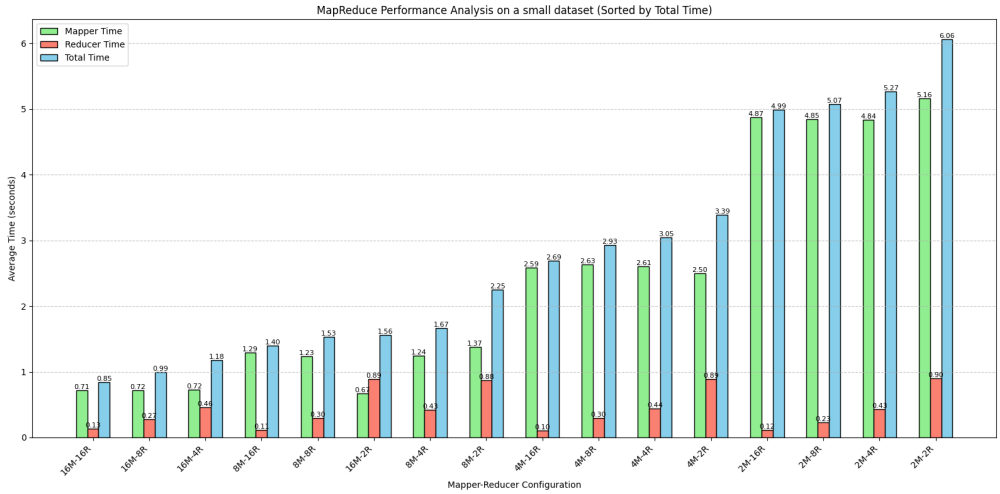

Figure 2: Scalability of PageRank with small datasets varying mappers and reducers.

Dataset Size Impact:

- Larger datasets increase the absolute time for both mappers and reducers, but the relative scaling behavior remains consistent.
- The performance gains from increasing mappers and reducers are proportionally similar across dataset sizes.

5 Discussion

5.1 Scalability of PageRank with Increasing Mappers and Reducers

The time complexity scales well with more mappers, however, the improvement is not linear due to communication overhead between mappers and reducers. Increasing reducers beyond a certain point (e.g., 8 reducers) provides diminishing returns.

6 Impact of Dataset Size on Performance

Larger datasets require more computation and communication, but the MapReduce framework effectively distributes the workload. The size of the data set significantly impacts the execution time of the PageRank algorithm:

6.1 Small Dataset

Execution times range from:

- 0.85 seconds (16 mappers, 16 reducers)
- 6.06 seconds (2 mappers, 2 reducers)

6.2 Large Dataset

Execution times range from:

- 3.62 seconds (16 mappers, 16 reducers)

- 24.59 seconds (2 mappers, 4 reducers)

Conclusion: Larger data sets require more computation and communication, leading to longer execution times. The relative speed-up from parallelization is more significant for larger datasets. There was an outlier when the cluster was handling 2 mappers and 4 reducers in the large dataset; however, even if the results differ from the norm, it still follows the linear trend with the amount of mappers and reducers on the time complexity scale.

7 Comparison with Word Count

The PageRank algorithm is more computationally intensive due to the iterative nature of the problem, whereas word count is a simpler aggregation task. The differences can be summarized as follows.

7.1 Computational Complexity

- **PageRank**: Involves iterative calculations and graph traversals, making it more computationally intensive.
- **Word Count**: A simpler aggregation task with no dependencies between words.

7.2 Communication Overhead

- **PageRank**: Requires communication between mappers and reducers to propagate rank contributions across the graph.
- **Word Count**: Minimal communication as each word can be processed independently.

7.3 Scalability

- **PageRank**: Scales well with more mappers and reducers, but is limited by the graph structure and communication overhead.
- **Word Count**: Scales almost linearly with more mappers and reducers due to its embarrassingly parallel nature.

Conclusion: PageRank is more complex and communication-heavy than word count, leading to different scalability characteristics.

8 Impact of Using Multiple Workers

Using multiple workers within the same compute node impacts performance in the following ways:

8.1 Advantages

- **Resource Utilization**: Multiple workers can utilize the node's CPU and memory more efficiently.
- **Reduced Communication Overhead**: Workers on the same node can share data faster than across nodes.

8.2 Disadvantages

- **Contention**: Workers may compete for shared resources (e.g., memory bandwidth), leading to diminishing returns.
- **Overhead**: Managing multiple workers at the same node introduces additional coordination overhead.

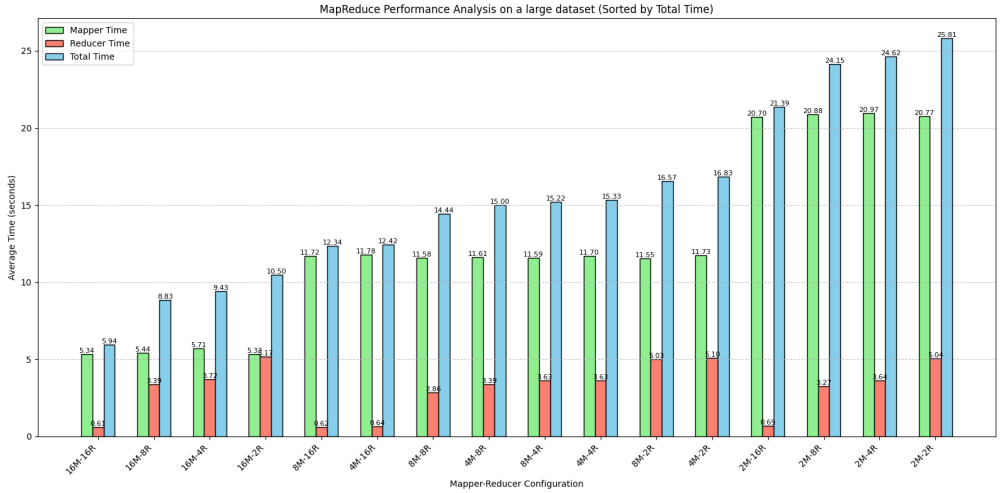

Figure 3: Scalability of PageRank with a large datasets within the same compute node

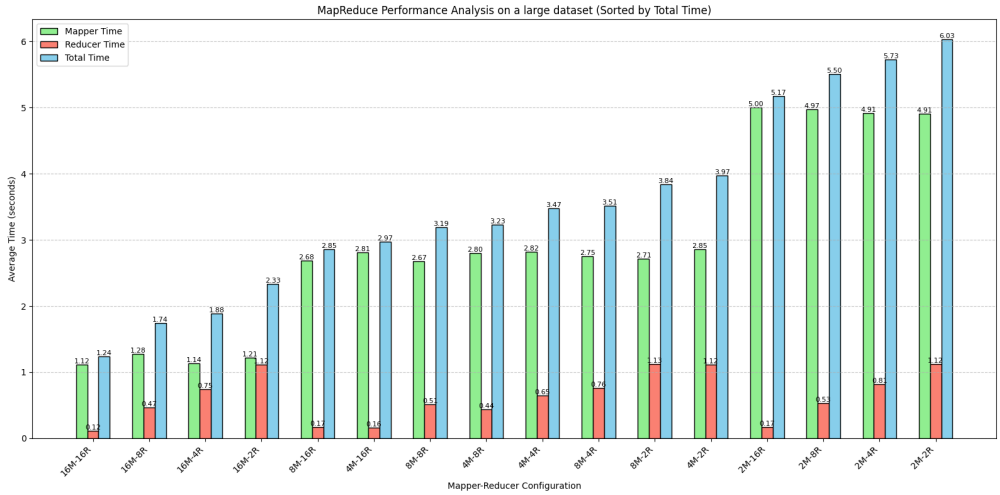

Figure 4: Scalability of PageRank with a small datasets within the same compute node

8.3 Evidence

For the time difference between the algorithm on a single compute node and in the cluster, the data showed a distinct trend.

Cluster vs. Single Node:

The cluster generally performs better than the single node due to distributed processing. However, the relative trends in performance improvement are similar in both setups.

8 Mappers and 8 Reducers:

In the cluster, the configuration with 8 mappers and 8 reducers achieves a total time difference of 7.58 seconds in the large dataset between the cluster and single compute node (averaged times). This is significantly better than the 2-2 configuration (1.66 seconds) and the 16-16 configuration (2.33 seconds), which were among the configurations that showcased the smallest time advantage when compared.

The 8-8 configuration shows the most significant improvement because it optimally balances parallelism and overhead in the cluster. This matches with the previous statement in the experiments section, which theorized that this was the case in the relation between mappers and reducers.

Small Dataset:

The same trend is observed for the small dataset, where the 8-8 configuration also demonstrates the most

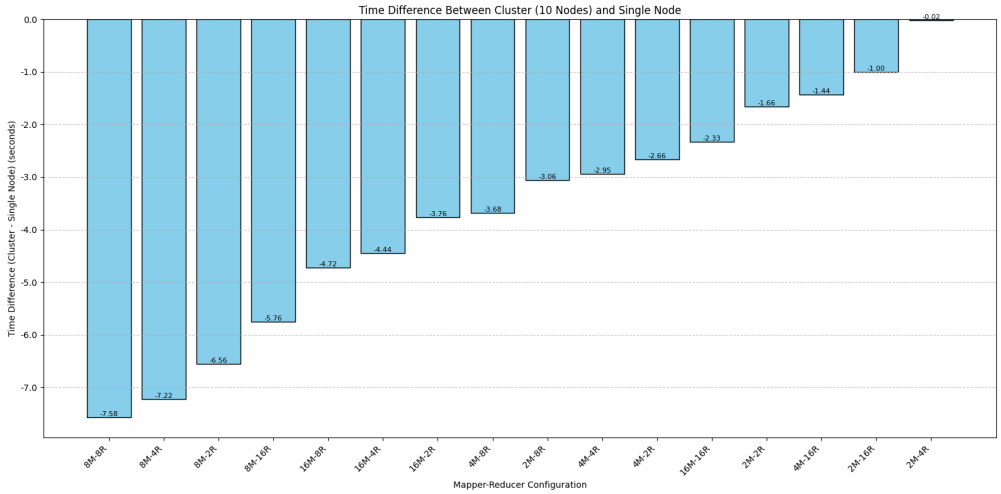

Figure 5: Scalability of PageRank with a large datasets within the same compute node

significant improvement. This consistency across datasets reinforces the conclusion that 8-8 is the optimal configuration for this setup.

9 Conclusion

The implementation of the simplified PageRank algorithm using MapReduce demonstrates the effectiveness of distributed computing for graph-based problems. The experiments show that the algorithm scales well with increasing resources but is limited by communication overhead and resource contention. These insights provide a foundation for optimizing distributed algorithms in real-world applications.