INF-3203: Advanced Distributed Systems

Assignment 1 - Map Reduce

Ine Arvola and Sera Madeleine Elstad

February 13, 2025

I. Introduction

This assignment explores the MapReduce model by implementing a simplified PageRank algorithm to evaluate scalability and performance. The goal is to compare configurations, computational complexity, and resource allocation.

II. Technical Background

A. MapReduce Programming model

The MapReduce framework, introduced by Dean and Ghemawat at Google in 2004, is a distributed framework for large-scale data processing. It consists of: a Map Function that processes input key-value pairs into intermediate key-value pairs and a Reduce Function that aggregates intermediate values to produce final output [1].

B. PageRank Algorithm

PageRank is a link analysis algorithm that ranks webpages based on the quality and quantity of incoming links. It assumes that a link from one page to another represents a vote of confidence, with the rank of a page u determined by the sum of the ranks of linking pages, weighted by their total outbound links [2]:

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)} \tag{1}$$

Here, B_u is the set of pages linking to u, and $L(v)$ is the number of outbound links from v.

Initially, all pages are assigned the same rank, 1. The MapReduce framework distributes computations by having the map function emit rank contributions to linked pages, while the reduce function aggregates these values. Iterations continue until ranks converge, reflecting link structure stability.

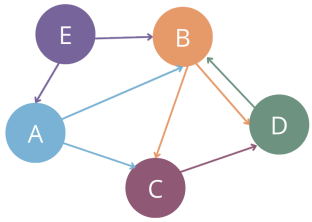

Fig. 1: Example of Link Graph

Figure 1 illustrates a directed graph with nodes (A, B, C, D, E) and arrows indicating links.

- A links to B and C.
- B links to C and D.
- C links to D.
- D links to B.
- E links to A and B but receives no incoming links.

Frequently cited pages, like B, gain more values as the rank distribution spreads using formula 1, while pages lacking incoming links, like E, become closer to zero.

This distributed method effectively ranks web content, scaling over big datasets by utilizing concurrent computation.

III. Design and Implementation

A. Mapper

The mapper processes input line by line, extracting the main page and its outgoing links. It emits `(page, "links:...")` to preserve the graph structure. Each page starts with a PageRank of 1.0, which is evenly distributed among outgoing links. For each link, it outputs `(linked_page, contribution)`. If no outgoing links exist, only the structure entry is emitted. The final output is `(page, PR, "links:...")`, retaining both PageRank and link structure.

B. Reducer

The reducer processes key-value pairs, separating PageRank contributions from structure data. Entries starting with `"links:"` are stored to retain outgoing links, while

rank contributions are summed. A union operation ensures all pages are included, assigning a default rank of 0.0 to those without incoming links. The output format is `(page, PR, "links:...")` if links exist; otherwise, only `(page, PR)` is emitted.

For this implementation there is only implemented one iteration of the reduce function, as the assignment text only expressed the need for one. As a result of this the algorithm implemented does not give an exact page rank.

C. Tests and experiments

For this assignment there is implemented two types of tests. One for functionality and one for performance. The functionality test is an extention of the sanity test given in pre-code. The performance test runs different mapper/reducer combinations on different sized datasets.

IV. Discussion and Results

The experiments was conducted on varying configurations of mappers and reducers, diffrent datasets, and different worker distributions across compute nodes.

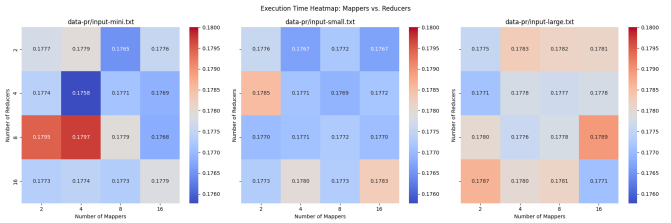

Fig. 2: Heatmaps for PageRank across different mapper and reducer configurations for varying dataset sizes

*1) **Scaling and Performance of PageRank with Varying Mappers and Reducers**:*

*a) **PageRank Complexity and Scaling**:* The performance of PageRank is heavily influenced by the dataset size and the balance between mappers and reducers. As shown in Figure 2, increasing the number of mappers and reducers generally improves execution time—but only to a certain point. An insufficient number of reducers can create bottlenecks by overloading individual tasks, whereas too many reducers introduce scheduling overhead, ultimately reducing efficiency. The optimal configuration varies depending on dataset size. While smaller datasets are more forgiving, extreme imbalances in mappers and reducers can still impact performance.

- For the mini dataset, execution times show noticeable variations due to the relatively high overhead associated with task coordination. This includes assigning workloads and transferring intermediate results between mappers and reducers. Since the computation time is relatively short, inefficiencies caused by overhead become more apparent.
- The small dataset demonstrates more stable execution times. However, when reducers are poorly allocated, inefficiencies still arise. The results suggest that while the impact of scheduling overhead is less severe for small datasets, it remains a factor.
- The large dataset highlights the biggest scaling challenges. The heatmaps indicate that increasing the number of reducers improves execution time up to around 8 reducers, after which performance begins to degrade due to scheduling overhead. Configurations with too few reducers combined with many mappers result in significant slowdowns. Choosing an optimal mapper-reducer balance is therefore crucial for efficient scaling and minimizing performance bottlenecks.

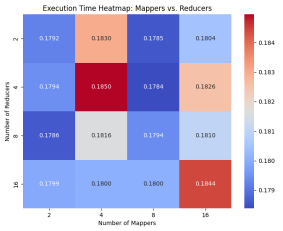

Fig. 3: Heatmap for word count.

*2) **Comparison with the Word Count Problem**:* PageRank differs from Word Count due to its iterative nature, requiring multiple iterations to converge, which increases overhead and sensitivity to mapper-reducer imbalances. Word Count, being a single-pass operation, avoids this complexity. However, since our experiment ran only one iteration of PageRank, its execution closely resembled Word Count, resulting in similar heatmaps. A full PageRank implementation would have significantly higher execution times due to iterative processing.

*3) **Impact of Multiple Workers on the Same Compute Node**:* The use of multiple workers within a single compute node improves CPU utilization but can also lead to resource contention. This explains why execution times for large datasets do not scale significantly better than for small datasets.

For small datasets, running multiple workers per node is efficient because the workload is relatively light. However, for larger datasets, assigning too many reducers to the same node creates contention for resources such as CPU, memory, and disk I/O. This results in diminishing returns when adding more reducers. Our results confirm that balancing not just mappers and reducers, but also the number of compute nodes used, is essential for achieving optimal performance.

*4) **Interpreting Our Experiments**:* Our experiments demonstrate that large datasets benefit from multiple mappers and re-

ducers, provided the configuration is well-balanced. However, achieving consistent results was challenging due to the shared cluster environment. Because other students were running workloads simultaneously, this likely introduced variability in execution times, making it harder to distinguish the impact of dataset size alone.

Despite this challenge, our findings suggest that:

- PageRank benefits from an optimal number of reducers—too few causes bottlenecks, while too many introduce overhead.
- Small datasets are more forgiving, but extreme configurations still reduce efficiency.
- Larger datasets highlight the impact of resource contention, emphasizing the importance of efficient workload distribution.
- Only one iteration of PageRank is not sufieccent to see the full results

By analyzing these results, we gain insight into how to optimize PageRank performance in a distributed computing environment. Our findings align with expectations from theoretical models of MapReduce, reinforcing the need for careful tuning of mappers, reducers, and compute node distribution.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," 2004.

[2] Contributors to Wikimedia projects, "PageRank - Wikipedia," Jan. 2025, [Online; accessed 3. Feb. 2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=PageRank&oldid=1272711370