Map Reduce Matrix Multiplication

Gerardo León Quintana

January 2024

Abstract

This paper introduces a novel approach to matrix multiplication, leveraging the MapReduce paradigm a widely employed technique for filtering and selecting data within extensive datasets. In this study, we apply the MapReduce framework to the domain of matrix multiplication, exploring its potential benefits in computational efficiency.

The experimentation is implemented using Python, utilizing the mrjob library to facilitate the integration of MapReduce. To ensure the rigor of our investigation, we conduct a comparative analysis of the execution times between MapReduce-based matrix multiplication and the conventional method. The matrices employed in this study are dynamically generated for each dimension explored, providing a comprehensive examination of the performance disparities between the two methodologies.

Through this comparative analysis, we aim to elucidate the efficiency gains or trade-offs associated with employing MapReduce in matrix multiplication, contributing valuable insights to the broader discourse on optimizing computational processes in data-intensive tasks.

You can find the whole implementation of the project in my Github.

1 Introduction

This experiment explores an innovative approach by applying the MapReduce paradigm, known for its prowess in handling vast datasets, to matrix multiplication.

Implemented in Python using the mrjob library, our study focuses on comparing the execution times of MapReduce-based matrix multiplication with the conventional method. The objective is to discern efficiency gains or trade-offs associated with integrating MapReduce into matrix multiplication.

2 Methodology

This experiment explores the efficiency of matrix multiplication using Python and the mrjob library, integrating the MapReduce paradigm. Randomly generated matrices serve as diverse test cases.

The MatrixMultiplication class extends MRJob, orchestrating the MapReduce process. The mapper assigns values to matrices A and B based on rows and columns. The **reducer-multiply** method computes the product of corresponding elements, emitting results as key-value pairs.

mapper-changeKey adjusts the key-value structure, and **reducer-sum** aggregates results by summing products associated with each key. Simultaneously, outputs are written to **Output.txt**.

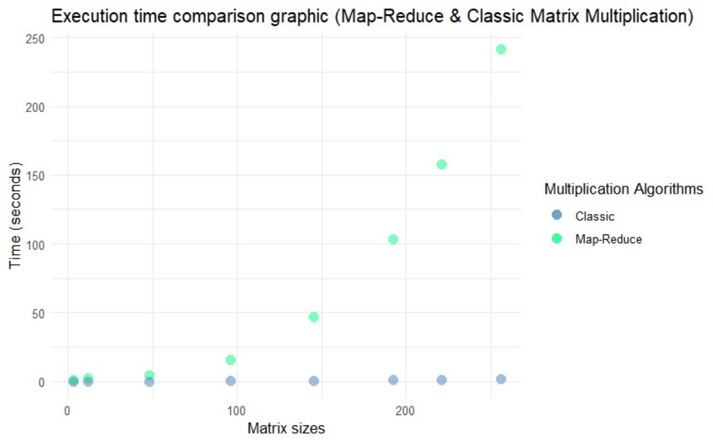
3 Experimentation

To replicate and validate the outcomes of our previous experimentation, we revisited the matrix multiplication approach, conducting a series of trials with varying input matrix sizes. Utilizing the same computational setup a laptop equipped with 16GB RAM and an AMD Ryzen 7 processor, operated this time through Pycharm.

The experiment aimed to evaluate the comparative performance between MapReduce matrix multiplication and classical sequential matrix multiplication. The focus remained on assessing execution time concerning different matrix sizes to affirm our prior findings.

A comprehensive graphical depiction encapsulates the comparative analysis between the two multiplication algorithms. The x-axis delineates the matrix sizes employed in our experiments, while the y-axis elegantly illustrates the corresponding execution times in milliseconds.

Figure 1:



Evident from the graphical representation is the overarching trend showcasing the superiority of the classical matrix multiplication in terms of overall performance.

4 Conclusion

In conclusion, our exploration of the matrix multiplication landscape has illuminated the intricate interplay of variables within routine mathematical operations, finely tuned for MapReduce implementation. The iterative process of experimenting with diverse matrix multiplication algorithms has not only enriched our understanding but also underscored the importance of continuous exploration. This journey serves as a testament to the imperative of always seeking improvement, encouraging us to move beyond initial successes and embrace a perpetual quest for refinement and innovation in algorithmic design.

5 Future work

The exploration of MapReduce-based matrix multiplication has opened avenues for future research aimed at advancing computational efficiency and addressing specific challenges within this framework. Key areas for further investigation include:

- Optimizing MapReduce for Sparse Matrices: Investigate strategies to enhance MapReduce efficiency specifically for sparse matrices. Sparse matrix representations and algorithms tailored to their characteristics could yield improvements in performance.
- Scalability in Distributed Environments: Extend the study to distributed computing environments, exploring the scalability of MapReduce for even larger datasets and matrices. Understanding the behavior of MapReduce in distributed systems will be crucial for real-world applications.
- Integration with Cloud Services: Evaluate the feasibility and performance benefits of integrating MapReduce-based matrix multiplication with cloud computing services. Assessing the impact of cloud resources on computation times could provide insights for practical deployment.