Distributed Matrix Multiplication: Scalability and Performance Analysis

Gabriel Felipe Bernal Pinto

January 12, 2025

Abstract

Matrix multiplication is a fundamental operation in numerous scientific and engineering applications. This paper explores distributed computing approaches for matrix multiplication using the Hazelcast framework, focusing on scalability and performance. We compare the distributed approach against basic and parallel implementations, testing with large matrices that exceed the memory capacity of a single machine. Our experiments demonstrate how distributed systems manage resource utilization and handle network overhead as matrix sizes increase, providing valuable insights into the trade-offs of distributed matrix computations.

1 Introduction

Matrix multiplication is a critical operation in various domains, including machine learning, numerical simulations, and data analytics. However, as matrix sizes grow, traditional methods struggle due to memory constraints and computational bottlenecks. Distributed computing provides a solution by distributing the workload across multiple nodes in a cluster, enabling efficient processing of large datasets. This paper examines the implementation of distributed matrix multiplication using Hazelcast, analyzing its scalability, network overhead, and resource utilization.

2 Methodology

Our implementation leverages the Hazelcast framework for distributed computation. The key components include:

- Distributed maps to store input matrices and results.
- A utility class to generate large matrices and store them in Hazelcast's distributed map.
- A matrix multiplication class that utilizes a thread pool to perform computations in parallel across nodes.

The matrix multiplication algorithm divides the computation by rows, assigning subsets of rows to different nodes. Synchronization and final aggregation are managed using Hazelcast's distributed data structures.

3 Experiments

3.1 Setup

Experiments were conducted using matrices of sizes ranging from 100×100 to 8000×8000, with node counts varying between 2, 4, and 8. Execution time, memory usage were recorded for each configuration. All of this experiments were performed on a PC (Intel(R) Core i7-7700K CPU @ 4.20 GHz, 16 GB of RAM).

3.2 Results

Table 1: Execution Time (ms) and Memory Usage per Node (bytes) for Distributed Matrix Multiplication

Matrix Size	Nodes	Execution Time (ms)	Memory/Node (bytes)
100	2	45	20000
100	4	13	10000
100	8	2	5000
500	2	116	500000
500	4	72	250000
500	8	50	125000
1000	2	1075	2000000
1000	4	743	1000000
1000	8	853	500000
2000	2	44051	8000000
2000	4	25975	4000000
2000	8	24731	2000000
4000	2	497730	32000000
4000	4	300112	16000000
4000	8	213438	8000000
8000	2	4819756	128000000
8000	4	2816075	64000000
8000	8	2411538	32000000

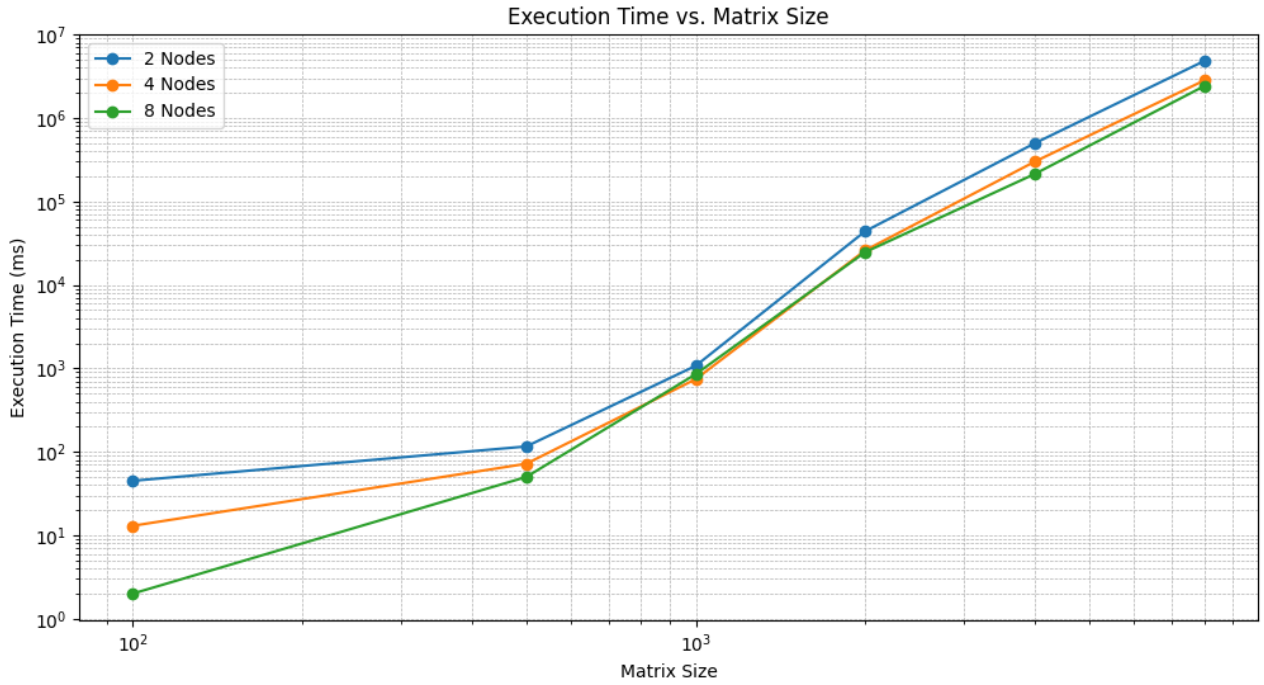

Figure 1: Comparison of Execution Times Across Nodes.

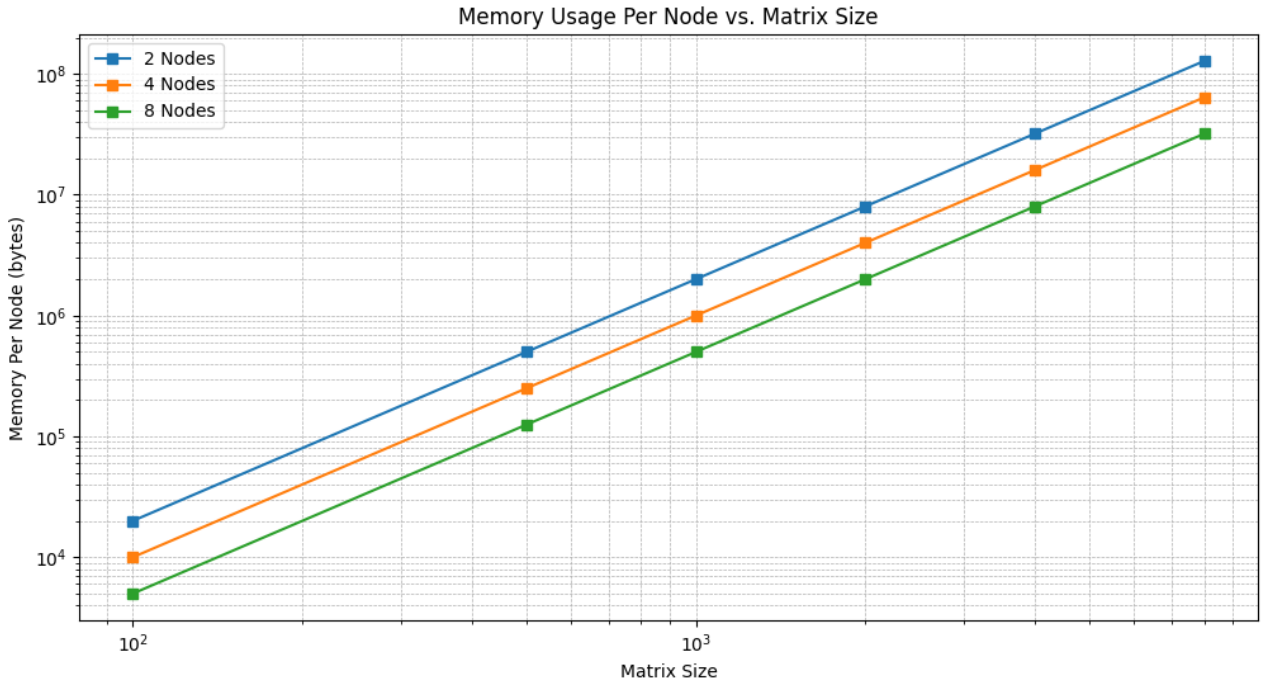

Figure 2: Comparison of Memory Use Across Nodes.

3.3 Comparison with Basic and Parallel Implementations

Table 2: Execution Time (ms) and Memory Usage for Basic Matrix Multiplication

Matrix Size	Execution Time (ms)	Memory Used (bytes)
128	32	172336
256	49	529424
512	306	2107408
1024	5742	8487104
2048	81010	33802480
4096	595972	136315904

Table 3: Execution Time (ms) and Memory Usage for Parallel Matrix Multiplication (4 Threads)

Matrix Size	Execution Time (ms)	Memory Used (bytes)
100	19.97	2000000
500	118.43	4000000
1000	2725.59	5000000
2000	34827.13	35000000
4000	330077.50	139000000
8000	3462121.28	601000000

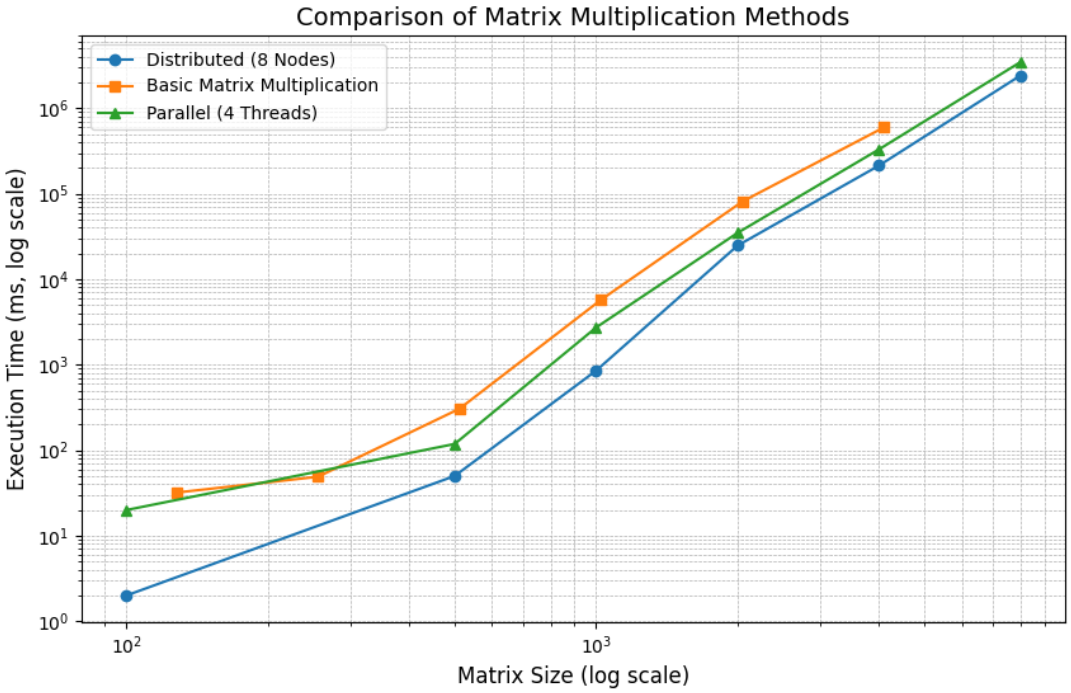

Figure 3: Comparison of Execution Times Across Implementations.

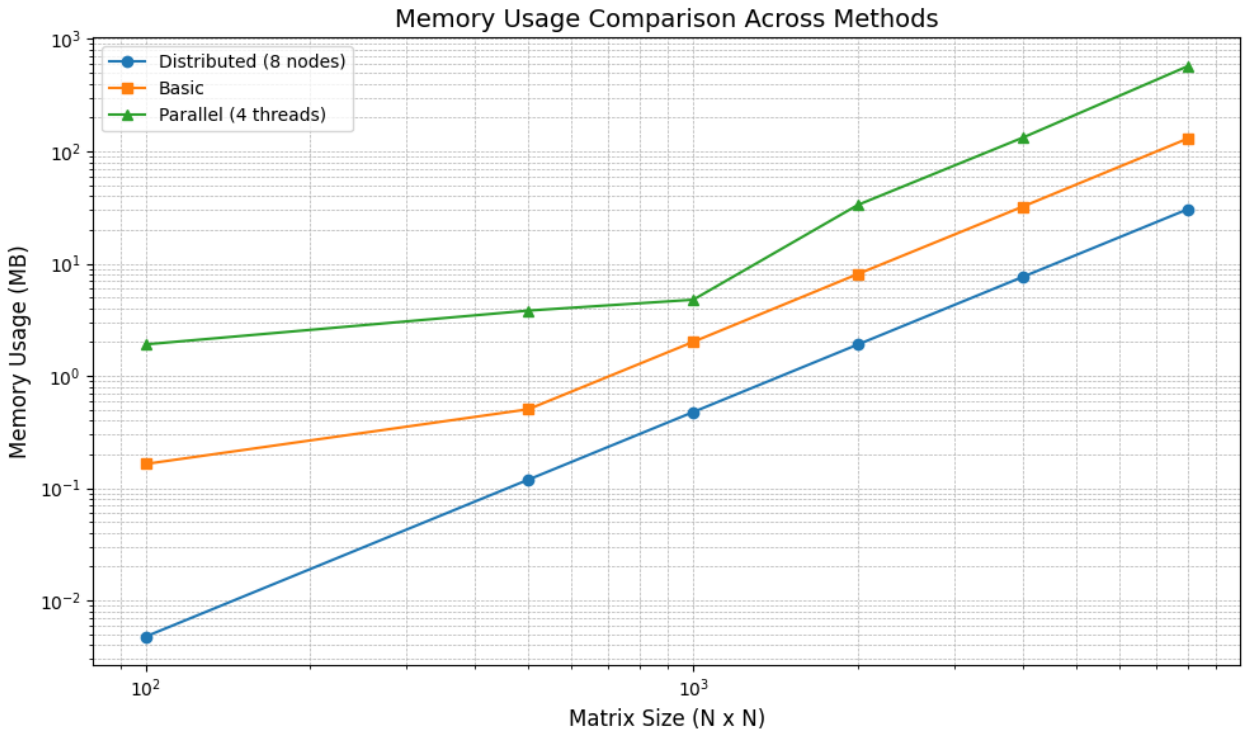

Figure 4: Comparison of Memory Use Across Implementations.

4 Conclusion

Our results highlight the scalability benefits of distributed matrix multiplication. As matrix sizes increase, distributed systems maintain reasonable performance by leveraging multiple nodes. In comparison, basic matrix multiplication exhibits significant performance degradation with increasing size, while parallel implementations improve performance but are limited by available threads and shared memory constraints. Distributed systems demonstrate the best scalability, particularly for extremely large matrices, though at the cost of increased network overhead and memory distribution.

The observed differences among the methods are primarily due to the underlying computational and resource-sharing strategies. Basic matrix multiplication operates sequentially, leading to high execution times as matrix size increases. Parallel implementations leverage multiple threads to divide

the workload, offering significant speedups but are constrained by the number of available cores and shared memory. Distributed computing, on the other hand, divides the computation across multiple nodes, allowing for efficient handling of memory-intensive tasks and maintaining performance even with extremely large matrices. However, this comes at the cost of network communication overhead and the need for careful synchronization between nodes.

5 Future Work

Future research will focus on optimizing network communication, exploring alternative data partitioning strategies, and testing on heterogeneous clusters. Incorporating GPU acceleration and improving fault tolerance mechanisms will also be considered to enhance the robustness and performance of the distributed matrix multiplication framework.

6 References

- Github Repository *Distributed Matrix Multiplication*.