Journal of **B**usiness **E**conomics
ISSN 2682- 6933 | E-ISSN 2705-9944
2025 Volume 07 Issue 02:20-28

OPTIMIZED TASK PARALLELISM FOR BIG DATA ANALYTICS IN A DISTRIBUTED SYSTEM

*R. Sibrikhan[1], T. Ketheesan[2], T. Jeyamugan[3], R. Nagulan[4]
[1,2] University of Jaffna, Sri Lanka
[3,4] University of Vavuniya, Sri Lanka
rmsifrikan6500@gmail.com

Abstract

As the scale of big data grows, efficiently processing and analyzing such vast amounts of information becomes increasingly challenging. Descriptive statistical analysis, crucial in many domains, often demands significant computation time when applied to large datasets. This research addresses the problem of slow execution in performing descriptive statistical analysis and generating graphs on large datasets, which is a bottleneck for timely insights and decision-making. The objective of this study is to optimize this process by leveraging task parallelism in a high-performance computing (hpc) environment. An optimized task-parallelism is introduced, which distributes computational tasks across multiple nodes using the message passing interface (mpi). Specifically, tasks such as calculating statistics and generating visual plots (histograms, percentiles, qq plots, cumulative distribution functions, etc.) Are divided among nodes in a distributed architecture, significantly reducing the overall processing time. This approach was evaluated using a large taxi trip dataset comprising approximately two million records, comparing the execution time of sequential and parallel processing methods. The results demonstrate that the parallel implementation achieves a 71.29% reduction in execution time, decreasing from 246.83 seconds (sequential) to 70.89 seconds (parallel). This significant improvement highlights the efficiency and scalability of the proposed task-parallel solution, offering valuable insights into enhancing big data analysis performance in distributed environments.

Keywords: *Big data analysis, task parallelism, high-performance computing (HPC), Message Passing Interface (MPI), descriptive statistics*

1. Introduction

In modern era, big data forms a critical force in the various fields like medicine, finance, transportation, and scientific studies. The unstopping proliferation of data quantity, velocity, and diversity (rooted in sensors, social media, business activities, and scientific instruments) increases accessibility to novel avenues of insight creation more than ever before. However, there are large and complex analysis problems involving the storage capability, processing speed, and decision-making in real-time given these scale and diversified information. Conventional sequential processing modes can no longer serve modern big data workloads to the extent perceived and required(Kambatla, Kollias, Kumar, & Grama ,2014). The main challenge associated with big data analysis regards the most efficient implementation of processing activities. In certain applications where time is crucial in the actual cost of fraud detection, traffic control or the prognosis of the disease, the cost of analysis can be wasted resources or negative consequences. With the demands on the analysis performance rising, especially in statistical computation and visualization(Cook, Lee, & Majumder, 2016), the sequential and single processor constrained execution are becoming ever more noticeable. In a bid to address these requirements, parallel computing has been proposed as a major remedy since it can be used to execute tasks in parallel in a variety of processors.

This parallelism improves speed, resource consumption and scalability thus increasing the degree to which it can be applied to big and complex data issues. It is in the field of multitasking computing that one finds a variety of models such as shared memory, distributed memory or hybrid arrangements. Among this range the Message Passing Interface (MPI) has proved to be especially useful in a distributed context (Li, Xu, Liu, Zhong, & Wang, 2024). Coordination and communication across processes of different machines is easily facilitated in MPI which makes it an apt choice when it comes to task distribution under big data applications (Ang, Ge, & Seng, 2020). In comparison to other mechanisms, like MapReduce(Cheng, Rao, Guo, & Zhou, 2014), MPI provides more control over the data exchange and most importantly synchronization-particularly beneficial when the task has interdependencies (Jeyaraj & Paul, 2022).

Sequential processing, in its turn, takes up one task at a time and therefore, cannot be scaled well and requires more time. Conversely, parallel processing breaks down operations into the smallest units and these can be executed at the same time which makes processing of data faster and more effective. Computationally intensive tasks(Chatterjee et al, 2011) like matrix operations, statistical analysis and visualizations, of which it is naturally possible to divide tasks are especially well suited to parallel processing(Cook et al., 2016). Descriptive statistical analysis is one of the most important tools of summarizing and interpreting large scientists. It also shows patterns and trends by calculating notions like the average, median, mode, standard, skew, and kurtosis. The use of distributed processing is necessary(Zafari, Larsson, & Tillenius, 2019) to expediently calculate these statistics in big data cases. Single processor based conventional techniques often face the problem of memory and computation constraints.

The limitations are alleviated in parallel algorithms, which compute the statistics locally and aggregate the findings by using the distributed data. Statistical charts and devoid of mathematics visualization(Wang & Lu, 2020) techniques (histograms, box plots, correlation matrices, etc.) contribute to a better explanation of statistical patterns. This paper analyses a task-parallel model, which relates to big data processing of high-performance or high-performance computing (HPC) settings(Paznikov & Kurnosov, 2024), intending to minimize execution time to provide descriptive statistical analysis and relevant visualizations(Cook et al., 2016) to enable efficient and effective insights within a shorter period. The research is carried out by implementing and benchmarking an iterative form of the algorithm and then creating/implementing a parallel form of the algorithm with MPI or any other framework(Klinkenberg, Samfass, Bader, Terboven, & Müller, 2020) that could be used instead. The parallel algorithm assigns work to a set of machines and uses load-balancing and low-communication techniques allowing the optimal use of the available machines(Schuchart, Tsugane, Gracia, & Sato, 2018). An evaluation that includes a comparison between the sequential and parallel implementation is given. The evaluation includes the analyses of the execution time on different data sizes, tracking resource consumption (CPU, memory, and network), and performing scalability tests in order to measure the performance when the number of machines grows.

2. Methodology

We have used an open source, proprietary data set the New York City (NYC) Taxi and Limousine Commission (TLC) trip record archive. The Data Set contains over two million individual trips records thus providing a full account of urban transit in the metropolis. The strong set of attributes in each record with the total size of the dataset reflects not only the potential and the limitation which are the hallmarks of computation and investigation in realistic big-data systems. The imported dataset was in its native format and needed only basic cleaning to have the completeness and assure correctness of information.

2.1. Traditional Sequential Method

The sequential algorithm was first of all implemented and run on a personal laptop to optimize the code, define baseline performance parameters in a controlled, resource-scarce environment. This strategy enabled the detection of bottlenecks and areas to optimize before an eventual scaling to a High-Performance Computing (HPC) system. The algorithm was then run on HPC platform to measure the improvements in its performance that were generated by parallel processing all the system's specification used here are presented in the Table 1.

Table 1: Systems Specification

System	Specification			
	Processor	Ram	Storage	Operating System
01.Personal Laptop	Intel i5 2nd Generation	8GB	128GB SSD	Windows 10 professional
02. HPC	Intel i7 9th Generation	32GB	250GB NVMI	Windows 10 professional

To determine baseline and ensure accuracy testing was carried out by means of a sequence of tests on a personal laptop. These analyses were later performed on a high-performance computing (HPC) system, which would allow one to compare the execution efficiency, performance assessment, and also decide upon the optimization strategy to use in generating a parallel implementation. Six variables trip distance, fare amount, total amount, tip amount, passenger count, and trip duration were used as significant in defining taxi trips and determining the overall experience and a thorough statistical analysis was applied to these variables.

The analysis started with descriptive statistics, mean, median and mode, to explain the central tendencies, and that the highest frequency of values was indicated by mode frequency. The data spread was estimated by using range, minimum and maximum values, whereas variance and standard deviation were used to measure variability. The supplying of information on asymmetry and sharpness of peaks, respective skewness and kurtosis, and the deeper look at the distribution was served by quartiles and the range between these two values (interquartile range, or IQR). It also involved advanced statistical measures to add to the analysis. The average deviations around the mean were obtained through mean absolute deviation (MAD) and the coefficient of variation (CV) provided a standardized variable of dispersion. Geometric and harmonic means were also computed to provide alternative views on the aspect of central tendency which is particularly useful in skewed or rate-based data.

2.2. Proposed Parallel Algorithm

This study was based on task-parallelism strategy which greatly increased the speed of the computing by dividing one large work into several small, independent subtasks that can be performed simultaneously using more than one processor. To support this parallelism, a distributed architecture was chosen, thus resulting in significant decreases in the time required in executions- a crucial aspect when it comes to examining big data due to its demands on computing power.

Communication between nodes was handled using Message Passing Interface (MPI). It has three interconnected computers through local-area network consisting of a Master node and two Child nodes. There was sharing of script files that could be accessed by all nodes and hostnames and IP addresses were configured to allow easy communication and used IP address are presented in the Table 2. The parallel processing environment has been set by the installation and proper configuration of MPI on every node and configurations of 3 nodes presented in the Table 3.

Table 1: Node and Assign IP address

Node	IP address
1. Master	192.162.1.1
2. Node1	192.162.1.2
3. Node2	192.162.1.3

Table 2: Host names and its configured slots

Node	Configured Slot
1. Master	1
2. Node1	1
3. Node2	1

The algorithm executed in parallel Atlas is divided into discrete, independent subtasks, including, especially, the computation of descriptive statistics and the generation of visualizations (histograms, percentile plots, Q-Q plots, CDF plots, correlation matrices). These Tasks can be undertaken simultaneously, hence maximizing the use of resources and making the execution processes less time consuming. The Master node performs this partitioning: it reads up the data, computes descriptive statistics and controls percentile-Plots. It then assigns the rest of the child tasks as same as in the figure 2, Child Node 1 gets Q-Q plot and correlation matrix and Child Node 2 gets the histogram and CDF plot. Subtasks performed by each child node separately and their results are reported to the Master node to be combined it shown clearly in the figure 3.

The approach towards task distribution is designed based on five principles. First task independence guarantees that any subtask can proceed without depending on others, so that actual parallel operation can actually occur, for example one node could make Q-Q plots as another makes histograms. Second, the balance of workloads is ensured by assigning two sub-tasks to each kid node and, thus, adjusting computational effort between the nodes. Third, a consensus is maintained by working with a single dataset and descriptive statistics at all nodes thus averting differences. Fourth, overhead required to communicate is low since only the passage of subtasks and product of work is communicated between nodes. Lastly, the centralized control architecture will be adopted whereby master node will orchestrate the task allocation, monitor the workflow and compile the output to consume coherent and effective deployment.

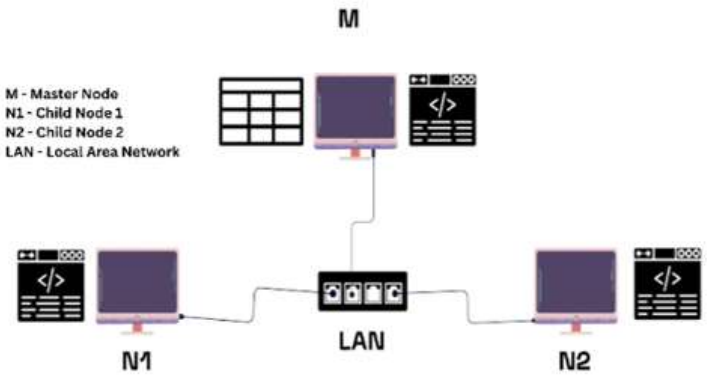

Figure 1: Experimental Setup for Parallel Execution

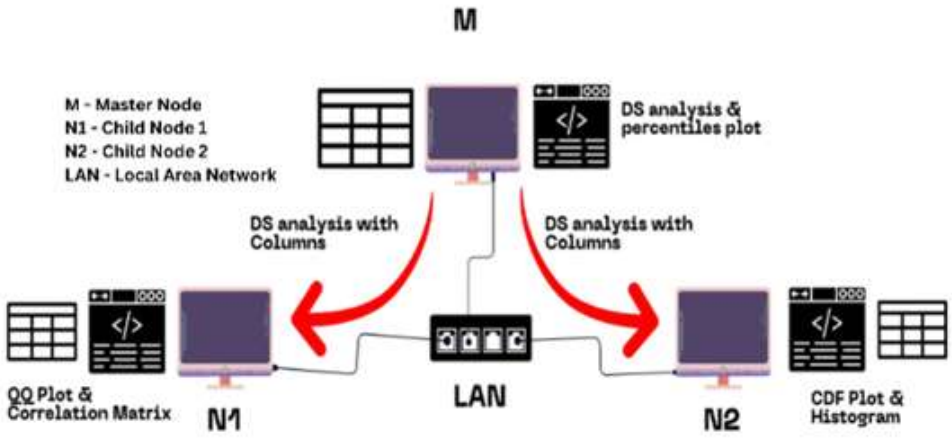

Figure 2: Master Executed & send the results to N1 & N2 and Both generate plots

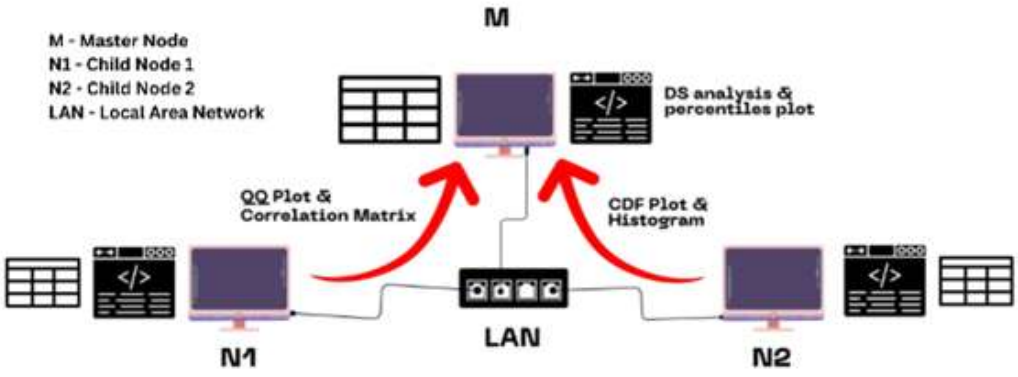

Figure 3: N1 & N2 Send the plots to Master Node

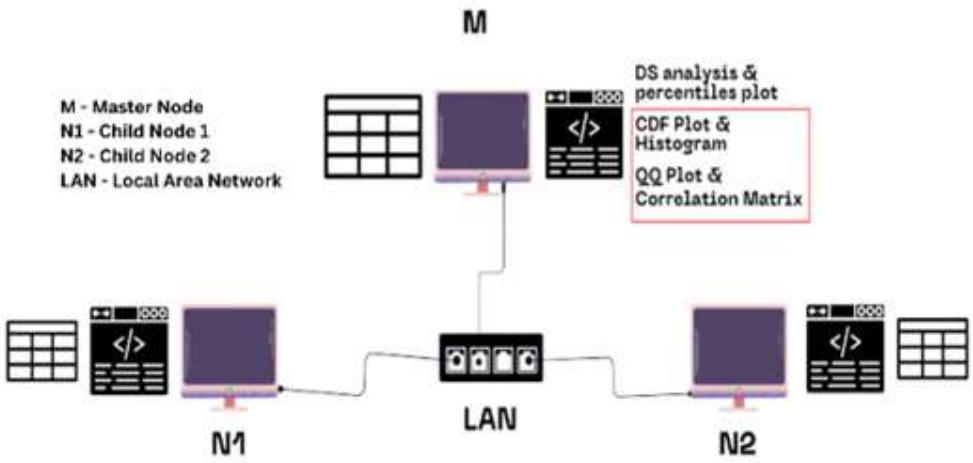

Figure 4: Master Node receives and saves the plots

In order to balance workload and reduce idleness, the Research often divide the tasks based on the plot types. The Algorithm has been presented as follows:

Step 1: Initialize MPI.
Step 2: Get the rank of the Communicator
Step 3: If rank = 0 (Master Node):
    3.1: Load Dataset.
    3.2: Compute the descriptive Statistics.
    3.3: Get the results and send it to Node1 and Node2
    3.4: Generate Percentile Plot.
    3.5: Receive Plots from Node1 and Node2.
    3.6: Store all the plots.
Step 4: If rank = 1 (Node1):
    4.1: Receive stats for QQ Plot and Correlation Matrix.
    4.2: Generate the Plots.
    4.3: Send Plots to Master Node.
Step 5: If rank = 2 (Node2):
    5.1: Receive stats for CDF Plot and Histogram.
    5.2: Generate the Plots.
    5.3: Send Plots to Master Node.
Step 6: End

3. Discussion and Results

The current study attempted to compare the relative performance of sequential and parallel processing paradigms, namely, in terms of execution time. When the analyses were performed on laptop one task after another, the total running time was 404.15 s. This is seen as a long time that highlights the drawbacks of sequential processing and the case used in large-sized data sets where each operation has to be completed before the other can be undertaken. Sequential method of execution is obviously linear and thus does not easily provide an opportunity to optimize over time against constrained hardware. A second successive test was performed on the master node; an efficient machine compared to the regular laptop and time of execution was lowered to 246.83s. Sequential method was time-consuming and less

efficient in dealing with large data although hardware configuration had a great impact on efficiency regarding the processing. However, the same tasks were performed in parallel on three nodes including the master node, 20% has been cut off the total execution time resulting in the entire job being completed within 70.89 s.

The result shows that large performance improvements can be done by parallel execution of tasks. Assigning self-sufficient operations to two or more nodes and their simultaneous implementation reduces time consuming, as well as underutilization making the best of every resource. The fall of 404.15s in personal laptop's sequential execution and 246.83s in efficient machine's sequential execution. In distributed parallel execution results to an increased execution time has been reduced to 70.89s of more than 71.29%. The observed results prove the efficiency of task parallelism when analyzing big data, and temporal efficiency varies in importance. Furthermore, the results confirm the purpose of the current study, which is to minimize the execution time using the parallel computing model and outlines the feasible merits of following the parallel architecture in practical data processing environments.

The speedup measure estimates the benefits that are obtained by using parallel processing compared to serial processing. In this case, a speedup of 3.48 implies that the parallel paradigm reduces results by about three times speeder than a sequential computation in the master node Complete Details with used system for this research including the execution time and speedup presented in the Table 4. The significant increase can serve as a reminder of how effective parallel processing can become, when dealing with extensive sets of data, as well as a second confirmation that parallel processing is ideally suited to improving computational efficiency in distributed computing systems. Specifically, the conclusion indicates the utility, in specific situations involving large amounts of input data, of parallel processing such as demonstrated by the value of division of work and parallel execution of work in continuing to create significant increase in speed of processing and speed of the system as a whole.

Table 3: Execution Time and Speedup Factor for Sequential vs. Parallel Processing

Execution Environment	Sequential Execution Time (seconds)	Parallel Execution Time (seconds)	Speedup
01.Personal Laptop	404.15	N/A	N/A
02.HPC	246.83	70.89	3.48

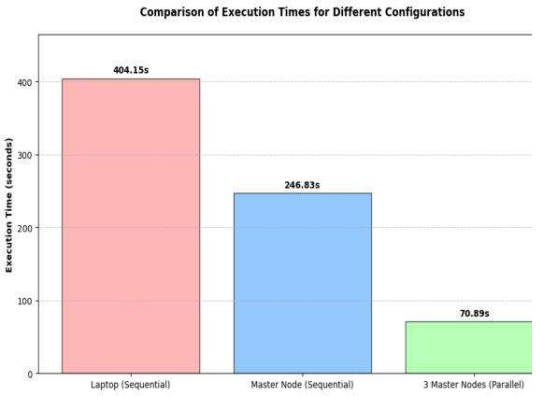

Figure 1: Execution time by configuration. Parallel processing on three nodes

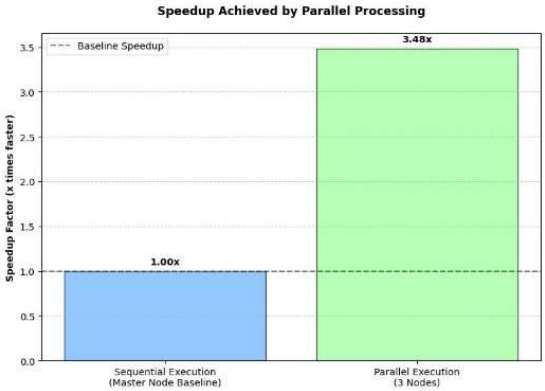

Figure 2: Speedup factor relative to master node sequential execution and parallel execution.

4. Conclusion

This research proves there are considerable benefits of parallel computing as applied to big-data analyses, especially in distributed computing environments using task parallelism. A direct comparison of the sequential and parallel processing methodologies using a common dataset has shown that the sequential processing by the master node took 246.83 seconds and the parallel processing by three nodes took 70.89 seconds. This decrease reinforces the fact that high performance can be gained by using parallelization. These results show that parallel processing can be used to process large datasets more efficiently, a concern of paramount concern in big-data environments where data sizes are quickly becoming prohibitive to single-node systems.

In particular, task parallelism can be viewed as a powerful process of spreading the load between several nodes and maximizing the use of resources, as well as reducing the execution time. The results are of specific importance to real-time data analysis and any other application based on speed and efficiency. Another fact that the study confirms is that parallel processing is the best way to use resources since it takes advantage of the computing capabilities of many nodes. Such scalability is essential to big-data analysis, where the processing can scale up with the volume of data without similarly increasing the amount of time spent on the computation. The study thus recommends that to achieve maximum performance in parallel computing environment, task distribution and communication between nodes ought to be optimized and reduced as much as possible. Despite the strong rationale of the benefits of parallel computing, there are certain limitations that have to be considered. Here we used only one dataset and few nodes, which are unlikely to reflect the magnitude of such an approach on more heterogeneous or larger deployments. The impacts of scaling the number of nodes, using parallel processing on a variety of datasets, and solving more complex tasks related to the analysis should be studied in the future.

Reference

Ang, K. L. M., Ge, F. L., & Seng, K. P. (2020). Big educational data analytics: Survey, architecture, and challenges. *IEEE Access, 8*, 116392–116414. https://doi.org/10.1109/ACCESS.2020.2994561

Chatterjee, S., Tasırlar, S. G., Budimlić, Z., Cavé, V., Chabbi, M., Grossman, M., … Yan, Y. (2011). Integrating asynchronous task parallelism with MPI. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*. Association for Computing Machinery. https://doi.org/10.5555/2042476.2042517

Cheng, D., Rao, J., Guo, Y., & Zhou, X. (2014). Improving MapReduce performance in heterogeneous environments with adaptive task tuning. In *Proceedings of the 15th International Middleware Conference (Middleware 2014)* (pp. 97–108). Association for Computing Machinery. https://doi.org/10.1145/2663165.2666089

Cook, D., Lee, E. K., & Majumder, M. (2016). Data visualization and statistical graphics in big data analysis. *Annual Review of Statistics and Its Application, 3*, 133–159. https://doi.org/10.1146/annurev-statistics-041715-033420

Jeyaraj, R., & Paul, A. (2022). Optimizing MapReduce task scheduling on virtualized heterogeneous environments using ant colony optimization. *IEEE Access, 10*, 55842–55855. https://doi.org/10.1109/ACCESS.2022.3176729

Kambatla, K., Kollias, G., Kumar, V., & Grama, A. (2014). Trends in big data analytics. *Journal of Parallel and Distributed Computing, 74*(7), 2561–2573. https://doi.org/10.1016/j.jpdc.2014.01.003

Klinkenberg, J., Samfass, P., Bader, M., Terboven, C., & Müller, M. S. (2020). CHAMELEON: Reactive load balancing for hybrid MPI+OpenMP task-parallel applications. *Journal of Parallel and Distributed Computing, 138*, 55–64. https://doi.org/10.1016/j.jpdc.2019.12.005

Li, Y., Xu, J., Liu, Y., Zhong, W., & Wang, F. (2024). MPI/OpenMP-based parallel solver for imprint forming simulation. *CMES – Computer Modeling in Engineering and Sciences, 140*(1), 461–483. https://doi.org/10.32604/cmes.2024.046467

Paznikov, A. A., & Kurnosov, M. G. (2024). MPI task mapping for multi-cluster HPC systems. In *E3S Web of Conferences* (Vol. 548, Article 03006). EDP Sciences. https://doi.org/10.1051/e3sconf/202454803006

Schuchart, J., Tsugane, K., Gracia, J., & Sato, M. (2018). The impact of taskyield on the design of tasks communicating through MPI. In *Lecture Notes in Computer Science* (Vol. 11128, pp. 3–17). Springer. https://doi.org/10.1007/978-3-319-98521-3_1

Wang, W., & Lu, C. (2020). Visualization analysis of big data research based on CiteSpace. *Soft Computing, 24*(11), 8173–8186. https://doi.org/10.1007/s00500-019-04384-7

Zafari, A., Larsson, E., & Tillenius, M. (2019). DuctTeip: An efficient programming model for distributed task-based parallel computing. *Parallel Computing, 90*, Article 102582. https://doi.org/10.1016/j.parco.2019.102582