Performance Improvement for Parallel Message Passing Applications

Nuha Al-Ameedi[1], Ahmed Fanfakh[2]
[1,2] Department of Computer ScienceCollege of Sciencefor Women University of Babylon, Hilla, Iraq
nuha.hussien.hadi@gmail.com, ahmed.fanfakh@uobabylon.edu.iq
Article History: 15 May 2021; Revised: 17 May 2021; Accepted: 28 May 2021; Published online: 20 June 2021

Abstract: The message passing interface (MPI) application is a parallel application that solves a problem by combining various communication routines with computation. Due to the synchronization between nodes, running this program on a heterogeneous platform would result in undesirable idle times. In other words, this may lead to a computational imbalance in the system. The increase in slack time between nodes, on the other hand, wastes the computing power of the computing nodes. A dynamic load balancing algorithm is proposed in this paper, which is applied to the parallel Jacobi application. It reduces slack times by using some information gathered from a parallel application dynamically. The obtained results speedup the execution time of an application by 15.40, 12.38, and 9.54 for problem sizes 8000, 6000, and 4000 respectively, compared to the serial execution time.
Keywords: Message Passing Applications, Heterogeneous cluster, Load Balancing

1. Introduction

Distributed parallel computing systems are a set of nodes connected through a communication network. It has some forms and infrastructure types like cluster, grid, and cloud, and so on. The local cluster is a set of nodes that are connected by the local area network. In almost all instances, computational capacities in nodes must be homogeneous. In certain cases, however, the variety of computers nowadays can be heterogeneous in the nodes' computing powers. However, enhancing cluster efficiency by spreading the working load appropriately across the computing resources is usually known as load balance [1]. A distributed system is a model that communicates and coordinates the behavior of networked components. The elements connect with each other to achieve a shared goal [2]. When parallel message passing programs running over this heterogeneous network are performed, the heterogeneity of node computing power generates an imbalanced workload. These occur when fast nodes suspend with the slowest nodes. However, the total runtime increases when idle times increase too [3].

When the heterogeneous cluster runs parallel distributed programs on the heterogeneous cluster, the variation in computing power results in an imbalanced workload. Speed-up is one of the most common metrics for calculating the parallel program performance. It is the ratio of an application's sequential execution time to its parallel execution time that solves the same problem. [4]. in this paper, a dynamic load balancing algorithm to recalculate the workload resulting from heterogeneity to reduce the produced idle time is proposed. It works depending on some gathered data from the parallel program. Moreover, three different communication routines are used to show how execution time can change corrodingly to the communication time.

The remainder of the paper is arranged as follows: some related works is presented in the Section 2. Section 3 explains load balancing in message passing systems. The proposed load balancing method for the MPI application is defined in Section 4. The experimental results are

shown in Section 5. The paper ends in section 6 with the conclusion and future work.

2 Related Works

The authors rely on many approaches, like statistical methods, mathematical models, and heuristic algorithms, to speed up the runtime of parallel applications. Recently, the primary area of concern in the distributed system is load balancing by moving the workload from heavily loaded nodes to lightly loaded ones. Therefore, increasing the throughput and minimizing the response and waiting time, it increases the overall performance of the computing device. Designing an efficient dynamic load-balancing algorithm is one of the most important issues in distributed systems because it contributes in improving the distributed system's performance [5]. Some of the interested works about the load balancing are in the following:

Authors in [6], proposed Load Balanced Fault Tolerant (LBFT) architecture using SOA includes a new dynamic load complementary algorithm, in addition to the fault-tolerant scheduling approach that has resulted in successful load balancing and fault tolerance. In [7], researchers proposed designing a fuzzy hybrid algorithm (FHA) to solve the problem of work schedules for efficiency purposes. The goal of the proposed system was to concurrently apply two algorithms to solve the same problem. To combine Q-Learning effectiveness with ant colony optimization to minimize the overall execution time. In [8] Mahato solved the load balancing problem by scheduling the colored Petri nets (CPNs) and evaluating the efficiency of the GTP system. In [9] had developed an improved grid Simulator method to heterogeneity conscious load balancing (EGHLB). The proposed algorithm estimates device parameters such as resource load, a Grid let's expected end time is an object that contains all the job-related information. Every arrival of the Grid let's balances the load by preparing to provide a new mechanism that prevents overloading of the resource and executes all the Grid let's within the allocated time limit on the heterogeneous resources. In [10] the method used for order choice resembled the optimal solution to resolve load balancing as an issue of decision-making in multiple parameters. Besides, an appropriate weighting mechanism is suggested to adjust the weights of the parameters considered to the present condition of the device and to the task features. . In [11], the authors suggested a negotiation-like process between the future WebSocket client and the target WebSocket server load balancing device to set the client on. They also announced that WebSocket links have various types of behavior, so this variation must be taken into account by an intelligent load balancer. In [12] the cuckoo search-ant colony was optimized via a hybrid algorithm. The method will be developed in an optimal schedule by clustering resources in light of their loads and making efficient use of resources to complete transactions on time.For the framework booking calculation, researchers in [13] suggested a load balancing scheme that would lead to the efficient provision of resources by conducting experiments to observe specific job parameters in a heterogeneous cluster. In a model for the provision of resources based on MLP (Multilayer Perceptron) and SVM (Support Vector Machine), the work history of computing heavy jobs was collected. The model's accuracy is measured and experimental results show that better performance of Multilayer Perceptron than of Support Vector Machine. In [14] the method used the load balancing Min-Min algorithm (LBMM). Initially, it meets the mission that has the least execution time and the commodity that generates it. In [15], the authors proposed a new Dynamic Load Balancing (JMADLB) Job Migration Algorithm to track parameters such as load processor and queue length. They had designated an Alea 2 Java simulator algorithm to migrate new jobs out of overload resources into underloaded resources and to test and evaluate algorithms. These algorithms were compared with other scheduling algorithms, such as First Come First Served (FCFS) and Earliest Deadline First (EDF). The dynamic load-balancing algorithm is proposed in this paper

to redistribute the workload based on some information gathering from iterative parallel applications executed over a cluster.

3 Load Balancing in Message Passing System

Distributed parallel system is a set of heterogeneous computing nodes linked by a high-speed network. It allows resources of the scheme shared by users from various locations across the communication network. This system is ideal when the computational workload is distributed over all of its computing nodes equally [16]. This prevents computing resources from being under-used and reduces the response time for work implemented on more highly loaded computing nodes. Method of computing power sharing is commonly referred to as "load balancing" for improving the efficiency of a distributed system by redistributing the workload between the computing nodes available. Load balancing aims to improve the efficiency of the system by redistributing the workload between computing nodes, thus improving the response time, the latency of communication, throughput, and utilization of resources [5].

The message passing routines use to run the parallel application over any distributed platform easily. The message passing program is composed of computation and communication parts. The task size computes before any task can communicate with others to solve the problem. However, if the size of all tasks not equivalent, then the workload is imbalanced, see Fig.1. Moreover, if all tasks have a similar work size, but execute over a heterogeneous platform, idle times will also produce according to the synchronization. The absence of idle time since all tasks must be equivalent in their size and execute over the homogenous parallel platform, see Fig.2. Therefore, his best message-passing program is the one that only has time for computation and communication with no slack times. The runtime of the parallel program is the execution time of the slower assignment [17].

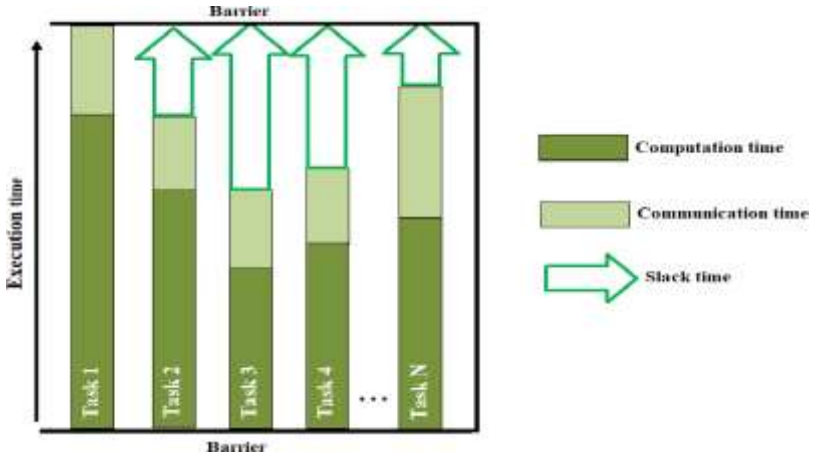


Fig.1. Load imbalance

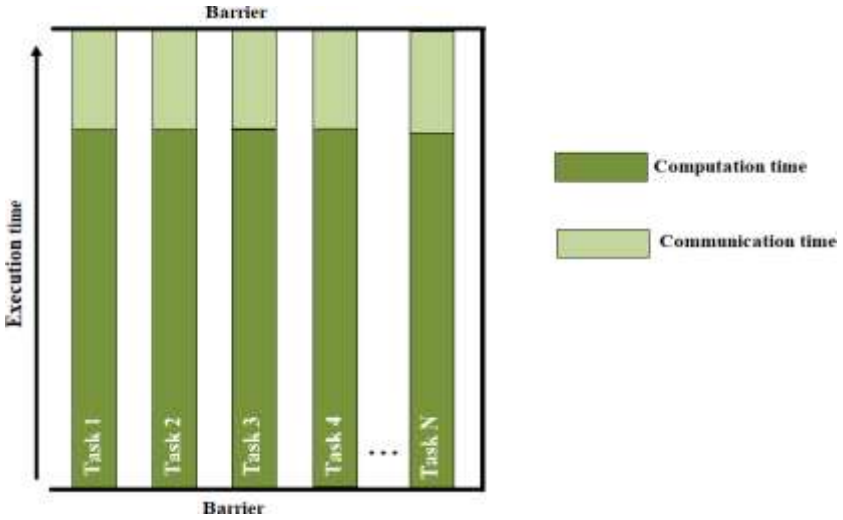


Fig.2. Load balance

Therefore, the execution time of parallel message-passing applications is calculated by the slower task. It has the greatest computing time and the lowest communication time, where slack times are not as in equation (1).

$$T_{parallel} = \max_{i=1,2,..N}(Tcp_i) + \min_{i=1,2,...N}(Tcm_i) \quad (1)$$

Whereas a result, the computation and communication times of node i are Tcp_i and Tcm_i respectively. This model calculates the parallel program execution time by calculating the measured computation time of the slower node. The communication time is also measured without slack time [3].

4 Proposed Load Balancing Method for MPI Application

This section aims to overcome an imbalanced computation problem when executing a message-passing application over a heterogeneous platform to speed up the overall running time. Each node differs in computing power from the other nodes in the heterogeneous cluster in the computing power. MPI programs are portable applications that work without changing a line in their code if they execute over different parallel hardware. Every program, therefore, consists of two components: computation and communication.

The proposed algorithm focuses on dynamically balancing the workload of the iterative method. It was applied to the distributed heterogeneous platforms when synchronous communication is used. The proposed algorithm works dynamically by gathering some information about the application at the running time to recompute the workload as a function of this information. After the first iteration, all computation times collect from all the nodes in the heterogeneous cluster and the maximum computation time is computed as follows:

$$Max\_Tcp = \max_{i=1,2,...,N}(Tcp_i) \quad (2)$$

Where, Tcp_i is the computation time of the node i and the number of nodes in the cluster

represented by np.

Whereas the load balancing algorithm is designed for synchronous iterative applications running on the heterogeneous cluster, the load balancing factor is computed to capture the changes in the workload between nodes. It is calculated for node i by dividing the maximum computation time by the computation time of node i as follows:

$$LF_i = Max\_Tcp / tcp\_all_i \quad (3)$$

Where, LF_i is the load balancing factor of node i, Max_Tcp is the maximum computation time, and tcp_all_i is the computation time of node i.

The new block per node is calculated by multiplying the load balancing factor from the equation (3) with the old block size as follows:

$$New\_bl_i = LF_i \cdot bl\_size \quad (4)$$

Where, New_bl_i is the new block of the node i.

Accordingly, the total number of new blocks is calculated by summing up all the blocks calculated in equation (4) as follows:

$$Tnbl = \sum_{i=1}^{np} New\_bl_i \quad (5)$$

By comparing the original size of the problem N with the total number of new blocks as in equation (5), there are some differences in the results based on the conversion from real to integer values.

$$Diff = abs(N - Tnbl) \quad (6)$$

The algorithm iteratively recomputes the differences in the equation (6) and redistributed it according to the load balancing factor calculated in the equation (3) as follows:

$$Diff_i = Diff \cdot \frac{LF_i}{np} \quad (7)$$

The flowchart in Fig. (3), adds $Diff_i$ to the new blocks if the total of the new blocks are less than the problem size N. Otherwise, this amount will subtract from all blocks. It stops the iterations when the result of the equation (6) is less than the number of nodes.

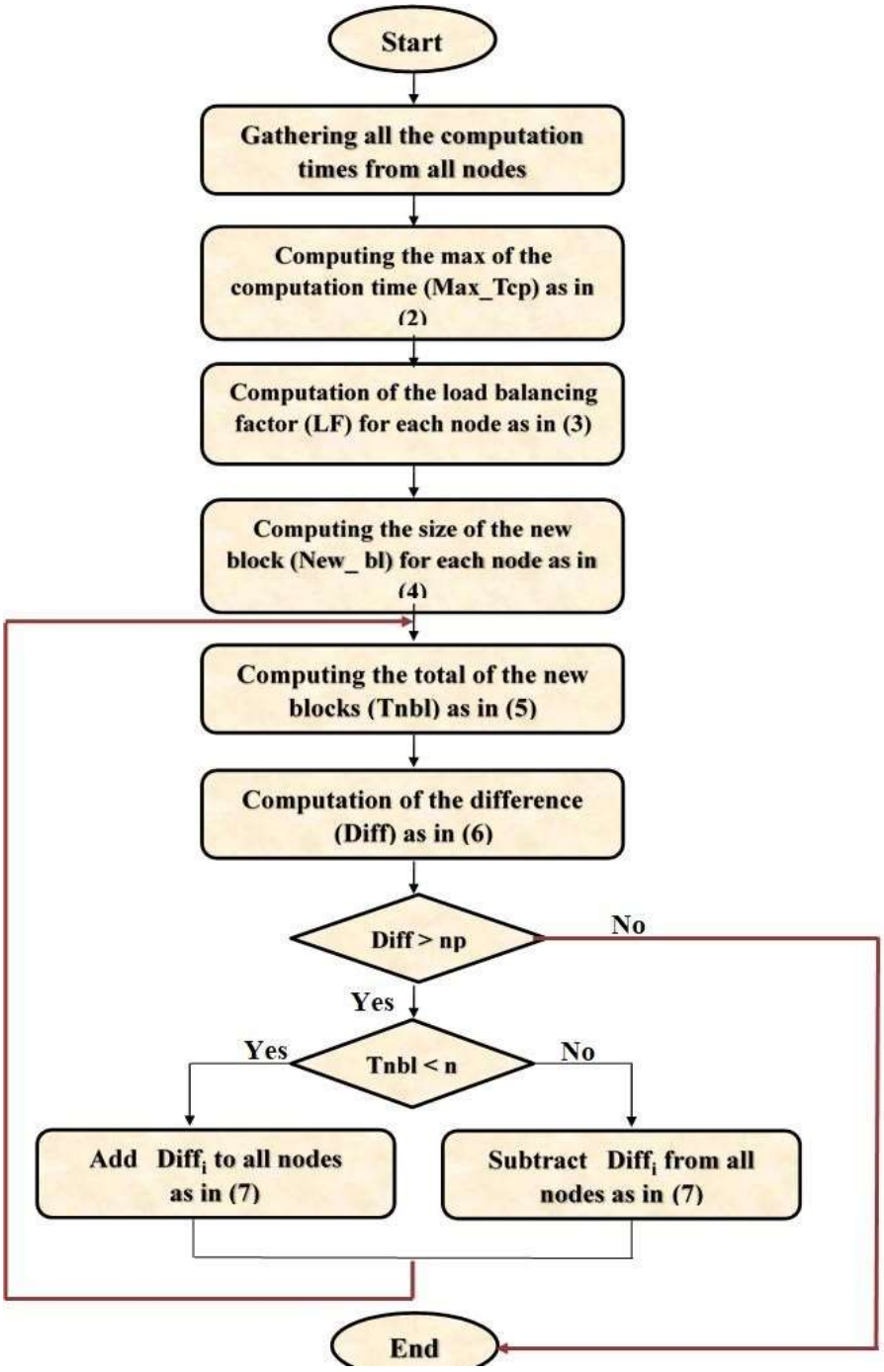


Fig. 3.The flowchart of the proposed load balancing method

5 Experiments

In this section, the proposed method is validated by next sub sections that show the experimental setting and the results.

5.1 Experiment Setting

This subsection uses the Linux operating system to program and execute the proposed method. Message passing interface uses to program and implements the parallel application and the

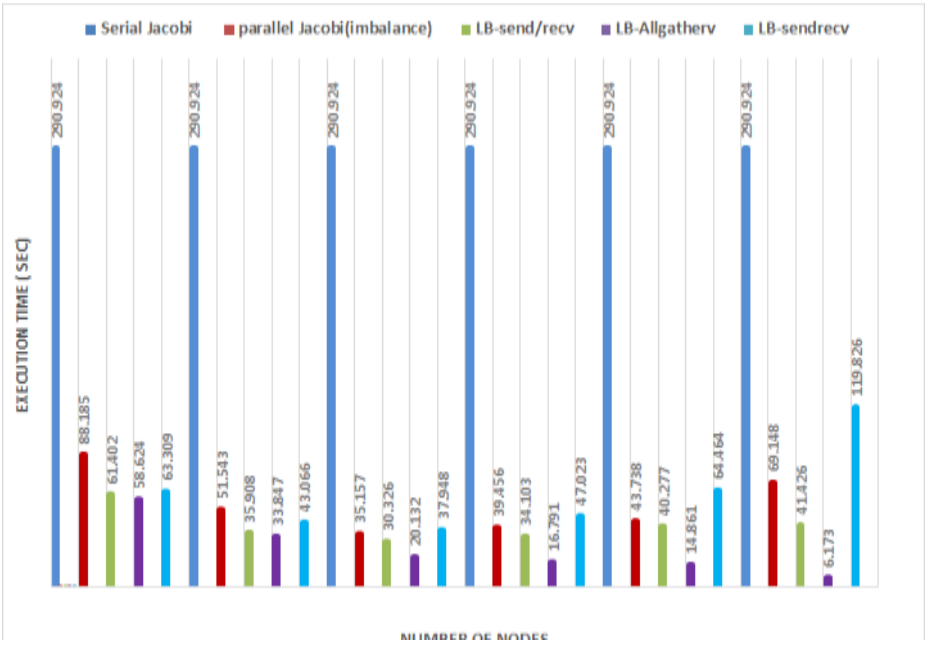
proposed method. The Jacobi method is used to validate the proposed load balancing algorithm. The Iterative Jacobi method was applied in two sizes in the matrix (4000×4000), (6000×6000), and (8000×8000). The residual value of convergence of the iterative Jacobi method was equal to (1×10-4) for all experiments. For implementing parallel heterogeneous clusters, the SimGrid/SMPI simulator is used. A heterogeneous cluster of four distinct node types selected to be used in the simulator. Every node in the cluster has different characteristics of computing powers (FLOPS) from others. Nodes are connected through a 10 Gbit/s bandwidth network with Ethernet. The detailed characteristics of the computing power of those four nodes are shown in Table 1.

Table 1. Heterogeneous Nodes' Characteristics

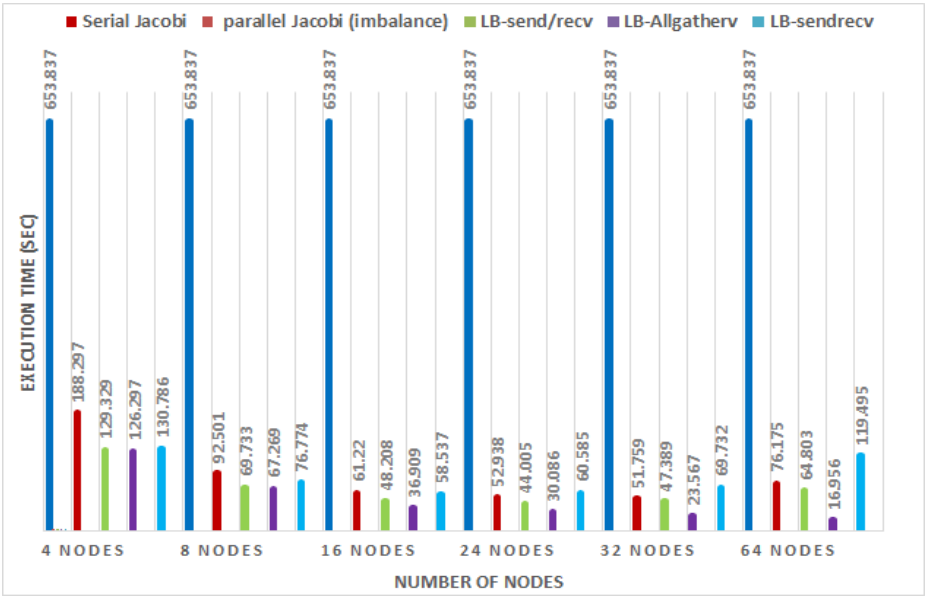
Node's number	Computational Power in GFlops	Processor freq. (GHz)
1	40	2.50
2	50	2.66
3	60	2.90
4	70	3.40

5.2 Results of the Experiment

The precision of the proposed load-balancing algorithm is verified in this subsection. Using Jacobi method, which solves the issue of a linear equation system, the load balance algorithm was implemented. A different number of nodes, 4, 8, 16, 24, 32, 64 nodes, were proposed to represent six scenarios. The initial execution time is captured for the parallel program Jacobi as in equation (1). Whereas the MPI has different types of communication routines, three types of these communication were implemented. Two point to point routines, which are standard Send/Recv and coupled SendRecv and allgatherv collective communications routine were used. Four parallel scenarios are compared to the serial version of iterative Jacobi method. The first scenario concerning the parallel Jacobi method that not solves the load balancing problem. This scenario used the standard send/recv communication routine. The other three parallel scenarios implement the load balancing algorithm each with different communication routine that demonstrated previously. The main goal of this scenario to study the suitability of these routines when using different number of nodes in a cluster. Fig.4 (a,b,c), clarifying the comparison of execution time of the four parallel scenarios to serial version of iterative Jacobi application. Three problem sizes 4000, 6000, and 8000 for an application are used. Besides, the proposed algorithm reduced the execution time of the Jacobi synchronous method by reducing the idle times. It also demonstrated the efficiency of the load balancing algorithm to improve performance the total execution time of the program by eliminating the waiting time that occurred when using synchronous communications.



(a) Resutt of problem size 4000



(b) Resutt of problem size 6000

Results of problem size 8000

Fig. 4. The execution time results

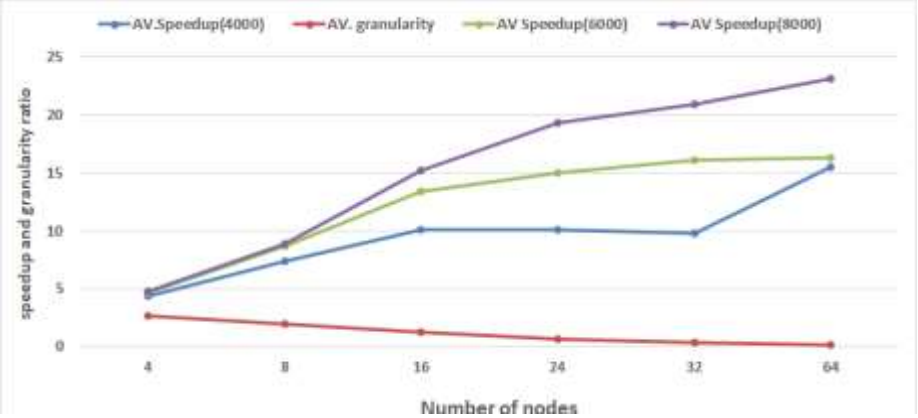


Fig. 5. The average of both speedup and granularity of all problem sizes (4000, 6000, and 8000)

The speedup ratio is calculated for all results that compare the ratio between serial execution time and the execution time of parallel application. Additionally, to study the ratio between computation and communication times, the granularity ratio was also calculated for all results. Fig. (5) show all these ratios and their relationship to the heterogeneous cluster's number of nodes. Acutely, the speedup ratio decreases when the number of nodes increases due to a decrease in computation. In other words, the decrease in the granularity ratio of the application will decrease its speedup ratio. Experiment results show that the proposed algorithm significantly balances the computations by speedup the execution time by 15.40, 12.38, and 9.54 times for problem sizes 8000, 6000, and 4000 respectively. Accordingly, the speedup ratio increased when the computation is increased too. The load balancing method that uses allgatherv communication routine is the best scenario in term of reducing the execution time of the serial version.

6 Conclusion

This paper introduces a dynamic load balancing algorithm for parallel iterative applications that improves performance and reduces execution time. To see how the execution changes by using various numbers of nodes, three different communication routines were applied. The heterogeneous cluster that runs the program is simulated using the SimGrid/SMPI simulator. Three different array sizes were used to evaluate the proposed Jacobi iterative approach for solving linear equations. The obtained result showed that the proposed load balancing algorithm improves the parallel application's execution time by 15.40, 12.38, and 9.54 for problem sizes of 8000, 6000, and 4000, respectively.

In the future, it will be important to improve the proposed approach to a variety of systems with the biggest challenges. Furthermore, the suggested approach will be investigated to see if energy consumption in parallel programs can be reduced while the application's execution time is reduced. Biggest problem sizes will be executed over many numbers of nodes in a cluster.

References

1. M. van Steen and A. S. Tanenbaum, “A brief introduction to distributed systems,” Computing, vol. 98, no. 10, pp. 967–1009, 2016, doi: 10.1007/s00607-016-0508-7.

2. Dharmik, R., & Sathe, S. (2018). A sender initiated dynamic and decentralized load balancing algorithm for computational grid environment using variable CPU usage. Int. J. Appl. Eng. Res, 13, 189–194.

3. Idrees, S. K., & Fanfakh, A. B. M. (2018). Performance and energy consumption prediction of randomly selected nodes in heterogeneous cluster. Communications in Computer and Information Science, 938(October 2018), 21–34.

4. Fanfakh, A. B. M. (2019). Predicting the Performance of MPI Applications over Different Grid Architectures. JOURNAL OF UNIVERSITY OF BABYLON for Pure and Applied Sciences, 27(1), 468–477.

5. Alam, M., Haidri, R. A., & Shahid, M. (2020). Resource-aware load balancing model for batch of tasks (BoT) with best fit migration policy on heterogeneous distributed computing systems. International Journal of Pervasive Computing and Communications, 16(2), 113–141.

6. Indhumathi, V., & Nasira, G. M. (2017). Service oriented architecture for load balancing with fault tolerant in grid computing. 2016 IEEE International Conference on Advances in Computer Applications, ICACA 2016, 313–317.

7. Hajoui, Y., Bouattane, O., Youssfi, M., & Illoussamen, E. (2018). New hybrid task scheduling algorithm with fuzzy logic controller in grid computing. International Journal of Advanced Computer Science and Applications, 9(8), 547–554.

8. Mahato, D. P. (2018). CPNs based reliability modeling for on-demand computing based transaction processing. ACM International Conference Proceeding Series, 6–9.

9. Patel, D. K., & Tripathy, C. R. (2018). On the design of an efficient load balancing mechanism on GridSim adapted to the computing environment of heterogeneity in both resources and networks. IET Networks, 7(6), 406–413.

10. Abdullah, A. M., Ali, H. A., & Haikal, A. Y. (2019). A reliable, TOPSIS-based multi-criteria, and hierarchical load balancing method for computational grid. Cluster Computing, 22(4), 1085–1106.

11. Alexeev, V. A., Domashnev, P. V., Lavrukhina, T. V., & Nazarkin, O. A. (2019). The design principles of intelligent load balancing for scalable websocket services used with grid computing. Procedia Computer Science, 150, 61–68.

12. Mahato, D. P., Sandhu, J. K., Singh, N. P., & Kaushal, V. (2019). On scheduling transaction in grid computing using cuckoo search-ant colony optimization considering load. Cluster Computing, 1–22.

13. Velayutham, V., & Chandrasekaran, S. (2019). Load balancing for effective resource provisioning in a heterogeneous cluster using machine learning. International Journal of Engineering and Advanced Technology, 8(6), 505–508.

14. Raushan, M., Sebastian, A. K., Apoorva, M. G., & Jayapandian, N. (2020). Advanced Load Balancing Min-Min Algorithm in Grid Computing. In Lecture Notes on Data Engineering and Communications Technologies (Vol. 31). Springer International Publishing.

15. WIDED, A., OKBA, K., & FATIMA, B. (2020). Load balancing with Job Migration Algorithm for improving performance on grid computing: Experimental Results. Adcaij: Advances in Distributed Computing and Artificial Intelligence Journal, 8(4), 5.

16. John, N. P., & Bindu, V. R. (2020). Prediction Mechanism – A Novel Approach for OverLoad Management in a Distributed Computing System. Procedia Computer Science, 171, 2097–2104.

17. Abdulazeez, Z. A., Fanfakh, A. B. M., & Alwan, E. H. (2020). Selecting Best CPU frequency for energy saving in cluster using genetic algorithm. IOP Conference Series: Materials Science and Engineering, 928(3).