ISSN 2063-5346

ENHANCED AFFINITY AWARE LOADBALANCING ALGORITHM

Varsha Thakur [1], Sunita Kushwaha [2]

[1]*Computer Science Govt NPG Science College,Raipur, India*

[2]*Computer Science, MATS University, Raipur,India*

Article History: Received: 12.05.2023 **Revised:** 25.05.2023 **Accepted:** 05.06.2023

Abstract

Parallel and distributed computing is in vogue as it proffers an approach to prevail over the limitations inflicted by the sequential computers. The issue is to dispense tasks in such a way that load among processors and resources should be in egalitarianism. Load balancing means providing optimal load to each processor in a multiple processor environment for which task is migrated from the over loaded processor to under loaded processor. Load balancing envisions by enhancing the design of scheduling process. This conveys the indispensability and reflects the main idea behind the work. In this work a new algorithm is designed in which affinity of task is taken into consideration. Simulation results show that most of the time newly proposed algorithm has better performance than other algorithms in terms of speedup, schedule length, efficiency, throughput, average utilization, unbalance parameter and load balance parameter.

Keywords: *Load balancing; Affinity; Multiprocessor; Schedule Length; Parallel Computing*

1. Introduction

Parallel computing conceding that multiple processors acclimatized simultaneously to execute a program. Parallel computing makes use of concurrently running processes that are an adherent of larger computation. Multifarious hardware and software gauntlets still exist while working with the parallel and distributed computing like Reliability, Scalability, Heterogeneity, Security, Job Scheduling, Synchronization, Load balancing and Communication delay [1-2]. Assortment of techniques and methodologies for assigning tasks is substantially reported in the literature survey. These techniques can be Task assignment approach, in which each task is scheduled to suitable processor so as to improve the performance, Load balancing approach which tries to equalize the load among processors by distributing each task among the processors and Load sharing approach in which the tasks are distributed in a way which simply attempts to assure that no processor should be idle.

2. Load Balancing

Load balancing resettled the work from the over loaded processor to under loaded processor [3-4]. The CPU loads can escalate as there is increase in a number of processes [5]. In the design of load balancing algorithms an appropriate load index play a pivot role. A load index

speculate the performance of a task. Load indexes can be measured by CPU length, memory availability , the context-switch rate, the system call rate, and CPU utilization [6]. Processor queue length, execution time and processes age [7-8] .

3. Problem Statement

The random arrival of a task in parallel architecture can cause some system to be heavily loaded while others may be idle or lightly loaded. Load redistribution [9] improves performance by transferring tasks. As long as there have been Parallel Architecture systems in which there are multiple processors, multiple tasks and various resources, there will be a basic problem of assigning tasks and resources to the processors by which equipoise of load should be maintained. The basic objective of load balancing can be expressed as follows: Let τ be the set of tasks $\{\tau_1, \tau_2, \tau_3, \tau_4, \ldots\ldots\ldots\ldots, \tau_m\}$ and β be the set of processors $\{\beta_1, \beta_2, \beta_3, \beta_4, \ldots\ldots\ldots\ldots, \beta_n\}$ on which elements of τ may be executed. Let a(j) be the set of tasks assigned to β_j processor and TLj is total execution time required by a processor β_j to finish the entire task in a(j). Hence, $TL_j = \sum_{j=1}^{n} tij$ where t_{ij} is i^{th} task on j^{th} processor, ti € a(j) for all task in a(j). TL_j defined as a load on node β_j where TL_j is total load on j^{th} processor. "Obtain a redistribution of tasks to processors in such a way that each processor contains an approximately equal amount of Load". The goal of load balancing problem is to find a redistribution of the tasks that minimizes the maximum load. A redistribution that balances the load of the processors will typically reduce the execution time and increase the overall performance. To combat the problem of Load balancing in parallel and distributed computing various algorithms have been proposed. Most of the Load balancing algorithms proposed are for tasks without considering properties of task like priority, affinity, I/O intensive or CPU intensive and dependency constraints. In order to deal with the aforesaid limitations, and to provide best load balancing solutions, some improved versions is needed. In the present work, affinity aware load balancing algorithm is proposed. Affinity helps to determine which tasks should be migrated from a heavily loaded processor to idle or lightly loaded processors. When affinity is contemplate in load balancing for data requirements, time and network bandwidth will consume less in order to deliver data that each task needed. Locality and load balancing are two competing goals in a parallel architecture [10].

4. Related Algorithm

After reviewing various Load balancing algorithms, some of the Load balancing algorithms are implemented in parallel architecture of processors.

4.1. Two state load balancing

In this Load balancing, a processor accepts new processes only if it is below the threshold value transfer the task to another node if is above the threshold value. [11].

4.2. Three state load balancing

This algorithm is based on two thershold value lower thershold and upper thershold. [12].

4.3. Four state load balancing

This algorithm is based on four states .There is a term called benefit function which used for decisiding the transfer of load. Load (L) is transferred only when load is greater than benefit function [12,13].

4.4. Cost Effective load balancing

The algorithm consists of three phases the information gathering phase, in which information is gathered about number of idle processors and load of each

processors.Then the decision phase in this decision is taken whether migration of task is cost effective or not. After this if saving is greater than cost redistribution phase is executed. In this phase, average workload and sum of active workload are calculated. Also, assignment policy is obtained [14-15].

4.5 Hierarchical load balancing

Processors are arranged in hierarchal organization. If parent load is overloaded it transfer task to children. If children node is not available then it transfers the load to its parent node [16].

5. Proposed Algorithm (EALBA)

In this paper, a new Enhanced affinity aware load balancing algorithm on parallel architecture of processors is proposed. If data already present in cache memory then transferring tthat related task is more efficient in case of shared memory multiprocessor [17]. In the proposed algorithm affinity helps for selecting task for transering.

5.1 Basic Assumptions

Non Preemptive Independent Tasks are generated randomly. Static deterministic approach has been considered, Affinity of task to particular processor is assumed some data of that task already exists in cache of that processor.

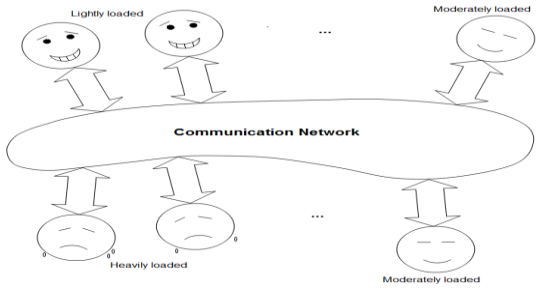

Figure 1: System without Load balance [6]

5.2 Algorithm

EALBA: This algorithm tries to balance the load among processors. Let $\beta_1, \beta_2, \beta_3, \beta_4, \ldots\ldots\ldots\ldots\ldots, \beta_m$ are the processors and $\tau_1, \tau_2, \tau_3, \tau_4, \ldots\ldots\ldots\ldots\ldots, \tau_n$ are the of processes or tasks. Tasks are created, input the number of tasks a[i] for each processor where $1 \leqslant i \leqslant n$. Assign the task randomly to the processor. Compute execution time as $tl1 = (\sum_{i=1}^{maxinst1} P[1].t[1].e[1])$ { $1 \leq i \leq maxinst1$}. $TL[m] = tl_1 + tl_2 \ldots\ldots\ldots\ldots + tl_{a1}$. $TL[1] = \sum_{i=1}^{a1} tl[i]$ where $1 \leq i \leq a1$. Double Threshold is calculated as multiplying some constant toaverage load (lower threshold is Φ_1 and upper threshold is Φ_2). Determine overloaded processor which has no affinity with the corresponding processor, only these processes can be transferred. Affinity less transferable process is taken in decreasing order of their execution times for transfer purpose. Let δ number of processes are transferred and number of processes present in maximum overloaded processor is $C\beta_i$ Then, $\beta_i \leftarrow C\beta_i - \delta$, $C\beta_k \leftarrow C\beta_k + \delta$. $TLmax1 \leftarrow TLmax1 - \gamma$, $TLmin1 \leftarrow TLmin1 + \gamma$. Repeat the steps for other over loaded processors Till max*(op,up) and Calculate performance parameters.

5.3. Complexity

. The complexity of an algorithm calculated in terms the time and complexity [18-19]. Time is major concern in our algorithm. So we have calculated time complexity. Problems can be categories as P class and NP class. Problem of P class has following characteristics: Solved in polynomial time, Solved by a deterministic algorithm. More specifically, they are problems that can be solved in time O(nk) for some constant k, where n is the size of the input to the problem [20]. The class NP consists of those problems that are "verifiable" in polynomial time. Verifiable means we are somehow given a "certificate" of a solution and then we could

verify that the certificate is correct in time polynomial in the size of the input to the problem [19]. If NP class problem solved by deterministic Turing Machine then exponential time is taken and if solved by a non-deterministic Turing machine then polynomial time is taken [19]. That is both the conditions are practically difficult to achieve.[20]. Example SAT is NP-complete. NP-complete problems are the subset of NP-hard problems. Example Circuit Satisfiability is NP-hard. The circuit satisfiability problem is the circuit analogue of SAT. Given a Boolean circuit C, we have to find an assignment to the variables that causes the circuit to output 1 [21,22].

O (EALBA) = 7* O(1) + 6*O(m)+ O(m*n)+O(n^2). So, O(EALBA) = O(n^2).

Where n is the number of processes. Best case that is the minimum time is taken when the number of processes and the number of processors equal to one.Therefore, time complexity of best case is O(1). If the number of processors is greater than number of processes then, the complexity will be O(m*n). Minimizing makespan is NP complete. But if we apply certain assumptions like limiting number of migration to a particular value then the problem of minimizing a makespan comes under the class of P class from NP class. By applying the limited number of migrations NPproblem can be transformed as P problems [23]. Since we have used deterministic algorithm in which steps are uniquely defined according to the definition of P-class problem. Further, the minimizing makespan is NP complete but applying the assumption of migrating limit minimize the makespan becomes P class.

6. Simulation

To realize the imbalance of load in parallel architecture of processors algorithms are implemented in Python (Numpy). Comparisons of various algorithms are done. These algorithms are implemented in Python (Initially implementation was done using OMP but these algorithms were showing the effect of parallelism that why python was selected). The performance of these load balancing algorithms are evaluated by considering various metrics like speedup, throughput, efficiency, average utilization, maximum schedule length and unbalance.

6.1. Performance Parameters

A parallel computing system should be viewed as a combination of parallel algorithm and the parallel computer on which it is implemented [24].

Speedup (S_n): Speedup is a ratio of execution time before changes and after changes. Change ≅Improvement ≅ Modification. [24].

Efficiency: While speedup measures how much faster a program runs on a parallel computer in comparison to a single processor. It does not measure whether the processors in that parallel computer are being used effectively or not. [24].

Throughput: Throughput is a measure of a number of tasks/processes that can be processed per time unit [25].

Schedule Length (Makespan): Schedule length is the overall execution time of all processes on all processors. It is also known as makespan [26].

Average Utilization: Average utilization is a summation of maximum times taken by the processors by schedule length is divided by the number of processors [27].

Unbalance: Unbalance is a maximum time taken by any processor minus minimum time taken by any processor divided by the average time taken by the processor [28].

Load balancing: Load balancing is the ratio of scheduling length and average execution time over all the processors [29].

6.2 Result Analysis

Performance Improvement			
Sno	**Parameter**	**Without Load Balancing**	**With Load Balancing EALBA**
1	Schedule length	147.45	72.81
2	Throughput	.171	0.34
3	Average Utilization	42 %	84.03%
4	Load balance Parameter	2.37	0.19
5	Unbalance Parameter	2.37	0.44
6	Speedup	-	1.99
7	Efficiency	-	.33

Table 1 Performance Improvement

TASK ID	ET	TASK ID	ET	TASK ID	ET	TASK ID	ET	TASK ID	ET	TASK ID	ET
*13	12.65										
*24	13.95										
*0	15.10										
1	12.83										
10	13.01	*16	16.22								
*23	16.22	21	13.01								
15	15.28	*14	13.95	2	13.77						
*6	14.89	3	15.10	*9	15.10	5	12.65				
7	15.28	*20	13.01	4	15.10	22	15.10	17	13.77		
8	16.22	12	14.49	11	16.22	*19	17.55	*18	16.22		

PROCESSOR 1	PROCESSOR 2	PROCESSOR 3	PROCESSOR 4	PROCESSOR 5	PROCESSOR 6
TET=145.47, TT=10	TET=86.20, TT=6	TET=60.20, TT=4	TET=45.31, TT=3	TET=29.99, TT=2	TET=0.0, TT=0

Figure 3 Before Applying Load Balancing

TASK ID	ET	TASK ID	ET	TASK ID	ET	TASK ID	ET	TASK ID	ET	TASK ID	ET
*6	14.89									1	12.83
*23	16.22	*16	16.22	2	13.77			21	13.01	10	13.01
*13	12.65	*14	13.95	*9	15.10	5	12.65	12	14.49	15	15.28
*24	13.95	3	15.10	4	15.10	22	15.10	17	13.77	7	15.28
*0	15.10	*20	13.01	11	16.22	*19	17.55	*18	16.22	8	16.22

PROCESSOR 1	PROCESSOR 2	PROCESSOR 3	PROCESSOR 4	PROCESSOR 5	PROCESSOR 6
TET=72.81, TT=5	TET=58.29, TT=4	TET=60.20, TT=4	TET=45.31, TT=3	TET=57.9 ,TT=4	TET=72.64,TT=5

Figure 4 After Applying Load Balancing (EALBA)

Experiments had been done to check the performance of proposed load balancing algorithm against some existing load balancing algorithms for parallel architecture. Performance of different load balancing algorithms is also obtained and compared for varying degree of processors and processes (tasks). As shown in the (Figure5-Figure11) increased the number of processes, keeping the number of processors fixed and observed the performance of these load balancing algorithms. Initially, 8 processes, 12 processes, 16 processes, 20 processes and 30 processes. Similarly we have changed the number of processors (4, 16, 32, 64) and fixed no of processes for various cases and average result was taken as shown in the Figure 12- Figure 18. Performance metrics in terms of schedule length, throughput, speedup, efficiency, average utilization, load balance parameter and unbalance parameter have been obtained and their relative values have been compared for various load balancing algorithms. From graphs it is clear that most of the times newly proposed Affinity Aware Load Balancing algorithm performs better than other existing algorithms.

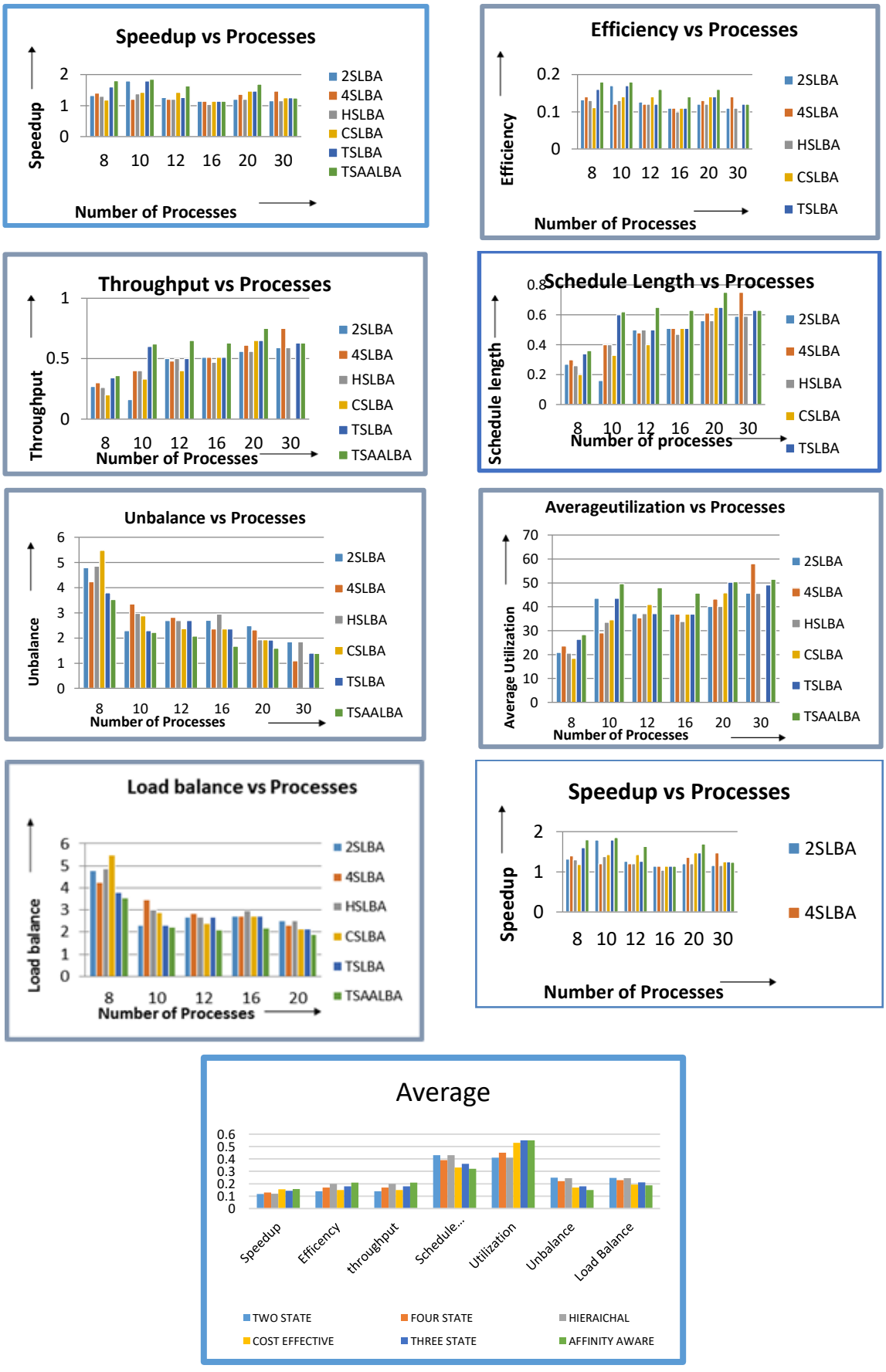

Figure 5 Result Analysis

7 Conclusions

As per simulation results most of the time newly proposed algorithm (EALBA) is found to work better as compared to other algorithms namely Two state load balancing algorithm, Three state load balancing algorithm, Four state load balancing algorithm, Hierarchal load balancing algorithm and Cost effective load balancing in terms of speedup, schedule length, efficiency, average utilization, unbalance parameter and load balance parameter. If the numbers of overloaded processors are more then, our proposed algorithm performs better. Cost effective load balancing algorithm searches the available idle processor then it finds the cost .Then it transfers the load according .If the number of idle processors is high, cost effective algorithm performs better. If there is no idle processor cost effective load balancing will not work at all. In Four state loads balancing algorithm load is transfer only when transferable load is less than the value returned by a function called benefit function. Many a times when although some processors are overloaded but their transferable load is not more than benefit function then load is not transferred. Therefore, Four state load balancing algorithm fails to work better in many conditions. Four state load balancing algorithm perform better in case of heterogeneous processors. Three state load balancing algorithm selects the process to be transferred randomly. Sometimes a processor which has more affinity with a particular processor on which it is residing, because of some reasons like its data being present in cache memory etc. may also be selected for transfer. In that case, processes are also deliberately transferred and then the result of Three state algorithm is not better. Hierarchal load balancing algorithm works on assumption that arrangement of processors are in hierarchal form. Performance of Hierarchal load balancing algorithm is better when child processor of overloaded processor is least loaded. In two states Load Balancing algorithm an overloaded processor will chose any under loaded processor randomly and transfer task to under loaded processor. But sometime it may make under loaded processor to overloaded processor. Future of this work includes: In this work, only independent tasks are considered for load balancing. In future dependent tasks with dynamic load balancing may also be taken up and Thermal aware load balancing algorithm in multiprocessor is to be explored. Effective use of energy and computational resources has become a matter of serious concern [30] so energy

References

[1] Dande, “Simulation of Multiprocessor System Scheduling”, Tampere University of Technology, Finland, pp. 18, 2011.

[2] J. L. Baer, ”Multiprocessing Systems”, IEEE transaction on Computers, vol. c- 25, no. 12, pp. 1271-1277, Dec. 1976.

[3] V. Thakur, S. Kumar, “Perspective study and analysis of parallel Architecture”, International journal of Computer Applications, vol. 148, pp. 22-25, 2016.

[4] P. K. Sinha, “Distributed Operating Systems concepts and design”, PHI Learning Private Limited, 2011, pp. 414, 367, 358,356.

[5] B. R. Vatsala, C.V. Raj, “CPU load–based countermeasure technique for intelligent DoS attack targeting firewalls”, Emerging Research in Electronics, Computer Science and Technology, Springer India, pp. 139–144, 2014.

[6] G. N. Shivaratri, P. Krueger, M. Singhal, ” Load Distributing for Locally Distributed Systems”, Computer, vol.25, pp. 31-44, 1992.

[7] P. Kanungo, ”Scheduling in Distributed Computing Environment Using Dynamic Load Balancing”, Anchor academic publishing, pp.35, 2016.

[8] P. Kanungo, "Measuring performance of dynamic load balancing algorithms in distributed computing applications", International Journal of Advanced Research in Computer and Communication Engineering, vol. 2, No. 10, Oct. 2013.

[9] Afzal, S., Kavitha, G. Load balancing in cloud computing – A hierarchical taxonomical classification. *J Cloud Comp* **8,** 22 (2019). https://doi.org/10.1186/s13677-019-0146-7

[10] S. W. Keckler, “The Importance of Locality in Scheduling and Load Balancing for Multiprocessors,” MIT concurrent VLSI architecture Memo 61, pp.1- 19, Feb.1994.

[11] R. Alonso, L. L. Cova, “Sharing jobs among Independently Owned Processors”, Technical Report CS-TR-200-88, Dept. of Computer Science, Princeton University, Eighths international conference on distributed computing, pp. 1-17, Nov.1987.

[12] W. Wang, X. Geng, Q. Wang, “Design of a dynamic load balancing model for multiprocessor systems”, IEEE, pp. 641–643, 2011.

[13] M. Haroon , M. Husain, “Analysis Of A Dynamic Load Balancing In Multiprocessor System”, International Journal of Computer science Engineering and Information Technology research, vol.3, no.1, pp. 143-148, Mar 2013.

[14] J. P. Ahrens and C. D. Hansen, “Cost-effective data parallel load balancing”, Technical Report TR-95-04-02, University of Washington, pp 1- 5, 1995

[15] S.L. Lee, C.T. Yang, S.S. Tseng, C.J. Tsai, “A cost-effective Scheduling with load balancing for multiprocessor systems”, IEEE, vo1. 1, pp. 302–309, 2000.

[16] J. Vladimir, “Load Balancing of irregular Parallel Applications on Heterogeneous Computing environments”, thesis, pp. 93- 100, 2012.

[17] G. Muneeswari, K. L. Shunmuganathan, “Agent Based Load Balancing Scheme using affinity Processor Scheduling for Multicore Architectures”, vol. 10, no. 8, pp. 12, 2011.

[18] M. S. Squillante, E. D. Lazowska, "Using processor-cache affinity information in shared- memory multiprocessor scheduling", IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 2, pp. 131-143, Apr. 1993.

[19] U. Agrawal,” Data Structure using C”, S.K. Katria and Sons, pp. 5-6, 2012.

[20] S. Lipschutz, “Data Structure”, Special Indian Edition, Tata McGraw Hill, 2006, pp.5.

[21] T.H.Cormen, C.E.Leiserson, R.L Rivest, and C.Stein, “Introduction to Algorithms”, Third edition Cambridge, Mass: MIT Press, 2001, pp. 850-855.

[22] https://cs.joensuu.fi/pages/whamalai/daa/npsession.pdf,revisted 3,May.,18.

[23] W. Tian, G. Li, X. Wang, Q. Xiong, Y. Jiang, “Transforming NP to P: An Approach to Solve NP Complete Problems”, arXiv: 1505.00058 [cs], pp. 1-5, Apr. 2015.

[24] V. Rajaraman and C.S.R Murthy,”Parallel Computers Architecture And programming”, PHI learning private limited, seventh edition, pp.7, 97, 339, 2009.

[25] J. P. Hayes, Computer Architecture and Organization, WCB/McGraw-Hill, Third Edition, 2009, pp.547-548.

[26] J. Cao, G. Bennett, and K. Zhang, "Direct execution simulation of load balancing algorithms with real workload distribution", Journal of Systems and Software, vol. 54, no. 3, pp. 227–237, Nov. 2000.

[27] A.Y. Zomaya, Y.H. Teh, "Observations on Using Genetic Algorithms for Dynamic Load-Balancing", IEEE Transaction on Parallel and Distributed systems, vol. 12, no. 9, pp 899-1002, Sep. 2001.

[28] V. Harsora, A.Shah, "A Modified Genetic Algorithm for Process Scheduling in Distributed System", IJCA Artificial Intelligence Techniques, pp. 36-40, 2011.

[29] S. Gupta, R. Rajak, G. K. Singh, S. Jain, "Review of Task Duplication Based (TDB) Scheduling Algorithms", The Smart Computing Review, Feb. 2015.

[30] M. A. Shahid, N. Islam, M. M. Alam, M. M. Su'ud and S. Musa, "A Comprehensive Study of Load Balancing Approaches in the Cloud Computing Environment and a Novel Fault Tolerance Approach," in IEEE Access, vol. 8, pp. 130500-130526, 2020, doi: 10.1109/ACCESS.2020.3009184.