Chapter 3

Load Balanced Transaction Scheduling

Load balanced transaction scheduling is an important issue in on-demand computing environments including grid system. This problem is known to be NP-hard and can be solved by using heuristic as well as any meta-heuristic method. We ponder over the problem of the load balanced transaction scheduling in on-demand computing system by using an Ant Colony Optimization for load balancing. The problem that we consider is to achieve good execution characteristics for a given set of transactions that has to be completed within their given deadline. We propose Load Balanced Transaction Scheduling based on Ant Colony Optimization (LBTS_ACO) [26]. We modify two meta-heuristic along with ACO and three heuristic scheduling algorithms for the purpose of comparison with our proposed algorithm. The results of the comparison show that the proposed algorithm provides better results for the load balanced transaction scheduling in the grid processing system. In this chapter, we model the problem and algorithm considering grid computing scenario which is also a one of the platforms for on-demand computing.

The objective is to find an appropriate schedule of the transactions from set T to the load balanced nodes from set N, that would optimize the performance criteria. Scheduling problem has several performance metrics. There are several standard performance metrics which are used in the scheduling. They include node utilization, system throughput, makespan [36, 34], mean response time [13], root-mean-square difference in queue lengths [13], mean system time, response time [35], mean waiting time in queue, mean idle time of the processor [13], task execution time [61], miss ratio [7] etc. In this

chapter, we use load, load deviation, node utilization, throughput, makespan, load balancing speedup, miss ratio, and makespan for the performance measurement.

3.1 Problem Formulation

We assume m number of the transactions and n number of grid nodes. Each transaction needs to be completed within their given deadline on one of the nodes. So, our aim is to find the maximum number of successful transactions that meet their given deadline.

Consider the set T of m transactions T_i $(i = 1,2,....,m)$ to be processed within their given deadline on the set of n nodes N_j $(j = 1,2,....,n)$. All the transactions can be processed on any of the n nodes. We assume that the completion time of each transaction, t_{T_i}, is independent of the node processing it. The formulations of some important parameters of the problem are given below.

The expected queue length $\mathscr{Q}_j$ at node N_j is given by the expression:

$$\mathscr{Q}_j = \sum_{i=1}^{m} T_{ij}.x_{ij}, \forall j = 1,\ldots,n. \tag{1}$$

where

$$x_{ij} = \begin{cases} 1 & \text{if } T_i \text{ is assigned to node } j \\ 0 & \text{otherwise} \end{cases}$$

Makespan [82] is the maximum needed time to complete the processing of all the transactions and can be calculated as

$$makespan(T) = \mathbf{max}\{t_{T_i}, \forall i = 1,\ldots,m\}. \tag{2}$$

Load L_j on the j^{th} node is calculated as

$$L_j = \frac{number\ of\ transactions\ in\ \mathscr{Q}_j}{service\ rate\ of\ N_j} \tag{3}$$

Thus, load on all nodes can be calculated as

$$L = \sum_{j=1}^{n} L_j \tag{4}$$

We compute the standard deviation of the load as

$$\sigma = \sqrt{\frac{1}{n} \sum_{j=1}^{n} (L_j - \bar{L})^2} \tag{5}$$

where σ is the standard deviation of the load with the same unit as load (%), n is the number of nodes, L_j is the load of the j^{th} node computed in Eq. (3), and $\bar{L}$ is the average load of all nodes. If the value of σ is small, the load of the entire system is balanced.

The node utilization [47] is calculated as the transaction completion time of each node by the makespan value.

$$U_j = \frac{completion\ time}{makespan} \tag{6}$$

Thus, the maximum load balance can be calculated as the sum of all the nodes‘ utilization divided by the total number of nodes.

$$\overline{U_j} = \frac{\sum_{j=1}^{n} U_j}{n} \tag{7}$$

For evaluation of the efficiency of the load distribution, load balancing speedup Θ is calculated as

$$speedup(\Theta) = \frac{t_{non\_balanced}}{t_{balanced}} \tag{8}$$

where $t_{non\_balanced}$ is the completion time without load balancing, and $t_{balanced}$ is the completion time after load balancing on the same set of nodes.

Transaction throughput of the system is calculated as

$$\mathscr{P} = \frac{Number\ of\ successful\ transactions}{total\ completion\ time}. \tag{9}$$

Suppose 90 transactions are completed in 60 time unit, and then the transaction throughput of the system will be $90/60$ or 1.5 transactions per unit time.

Miss Ratio is also used to measure the performance of the transaction oriented systems. It is calculated as

$$Miss\ Ratio(T) = \frac{Number\ of\ aborted\ transactions}{Total\ number\ of\ transactions} 100\% \tag{10}$$

With the **Eqs.** (1) to (10) taken into account, the transaction processing model for load balanced transaction scheduling is formulated as follows:

$$\begin{aligned} &\text{Minimize} \quad L \\ &\text{subject to} \quad proc_j \leq makespan(T), \forall j = 1, \ldots, n. \end{aligned} \tag{11}$$

where the constraint 11 states the processing time $(proc_j)$ of all the transactions on node N_j should lie within its makespan.

Minimization of makespan of a set of the transactions will maximize the number of committed transactions. The maximization of successful transactions will result in maximizing the throughput of the system (see **Eq.** (9)). If throughput is increased, it means the number of committed transactions are said to be increased. Because the maximum number of the transactions are getting chances to access the nodes successfully within their deadline. Thus, the minimization of the makespan becomes a prime concern while enhancing the performance the system.

$$\begin{aligned} &\text{Minimize} \quad makespan(T) \\ &\text{subject to} \quad \sum_{j=1}^{n} x_{ij} = 1, \forall i = 1, \ldots, m \end{aligned} \tag{12}$$

where constraint 12 states that each transaction should be assigned to exactly one node.

Minimization of makespan also minimizes the miss ratio of the transactions which is also a major concern for the better performance of the grid transaction processing system.

Therefore, we propose the load balanced scheduling algorithm with the objective functions mentioned in **Eq.** (11) and **Eq.** (12).

3.2 The Proposed Ant Colony Optimization for Balanced Transaction Scheduling

This section presents a transaction processing algorithm LBTS_ACO based on Ant Colony Optimization [35, 36] to solve the load balanced transaction scheduling problem.

3.2.1 The proposed algorithm: LBTS_ACO

The LBTS_ACO algorithm (**Algorithm 1**) is used to find the optimal solution such that the less loaded nodes can be selected for scheduling the transactions. The LBTS_ACO algorithm informally can be imagined as the interplay of three procedures: the first procedure constructs the solutions. The second procedure updates the pheromone_load value trailed by the ants. While the third procedure updates the global solution. We use the term pheromone_load in this chapter to connect the load concept. We need to select those nodes which have the minimum load. Therefore, the node with the less value of pheromone_load is highly preferable to be selected. Because our motive is to apply load balanced scheduling for the transactions.

- **ConstructAntsSolutions** manages a colony of those transactions, which concurrently and asynchronously visit adjacent states of the problem. They apply a stochastic local decision policy while moving with the use of pheromone_load trails and heuristic information. In this way, solutions to the optimization problem are built incrementally.

- **UpdatePheromones** process modifies the pheromone_load trails. The trail's value either increases, as the transactions deposit pheromone_load on the components or the connections they use, or decreases, due to pheromone_load evaporation.
- **DaemonActions** procedure is implemented as centralized actions. The activation of a local optimization procedure or the global information collection is the example of this procedure that decides whether additional pheromone_load is useful or not.

The output of the LBTS_ACO algorithm forms a scheduling queue.

This algorithm is a stochastic search procedure of nodes carrying the minimum or the null load. The pheromone_load model [54] is the central component of this algorithm which is used to sample the search space probabilistically. The process is a Combinatorial Optimization problem.

TABLE 3.1: Parameters of LBTS_ACO

Parameters	Value
α	0.5
β	0.5
ρ	0.2
d	0.25

Definition 3.2.1. *Given load L_j on the sender node N_j, and two random nodes N_{r1} and N_{r2} with loads L_{r1} and L_{r2} respectively, then*

$$\phi(L_j, L_{r1}, L_{r2}) = \begin{cases} \delta_{jr1r2} & \text{if } L_{r1} < L_{r2}, \\ \delta_{jr2r1} & \text{if } L_{r2} < L_{r1}, \\ 0 & \text{otherwise.} \end{cases}$$

where $\delta_{jr1r2} = |\frac{L_{r1}-L_j}{L_{r2}-L_j}|$, $\delta_{jr2r1} = |\frac{L_{r2}-L_j}{L_{r1}-L_j}|$, $r1, r2 \in N$, and $\forall j = 1, \ldots, n$. At least one of the δ_{jr1r2} and δ_{jr2r1} should lie in the range $[0.0, 1.5]$.

Here we select the value of δ from the range $[0, 1.5]$. Suppose, there are four nodes N_1 (as the sender), N_2, N_3, and N_4 as receivers having the load on them as 8, 16, 10, and 5 respectively. If we select two receiver nodes randomly as N_2 and N_3, we calculate $\delta_{132} = \frac{10-8}{16-8} = 0.25$. Here we see that $\delta_{132} = 0.25$ which lies in the range $[0, 1.5]$. This means this pair of receivers is suitable at the particular iteration. Then we select the node N_2. If neither δ_{jr1r2} nor δ_{jr2r1} lie in the defined range $[0, 1.5]$, it means the load difference

between the sender node and the receiver node is very high. So the node is not suitable to be selected.

Definition 3.2.2. *If Lp_j be the pheromone_load value on the j^{th} node, K is the iteration number, and ρ be the given evaporation rate of the pheromone_load of an ant active on the node, then Lp_j can be updated using the evaporation rate ρ as $Lp_j(K+1) \leftarrow (1-\rho).Lp_j(K) + \frac{\rho}{K}.\sum_{k=1}^{K-1} Lp_j, \forall k \in K$.*

The parameter $\rho \in (0,1]$ is the evaporation rate. It has the function of uniformly decreasing all the pheromone_load values. From a practical point of view, pheromone_load evaporation is needed to avoid a too rapid convergence of the algorithm toward a sub-optimal region [54]. The value of ρ is given in **TABLE 3.1**. We assume that the initial value of pheromone_load is $\frac{d}{log(k+1)}$, where d is constant. The value of d is given in **TABLE 3.1** and K is the maximum number of iteration.

Definition 3.2.3. *If ρ be the given evaporation rate of the pheromone_load of an ant active on the node selected by $\delta(j,r1,r2)$, the quality of the selected node is $\eta(N_j)$, where $\eta(N_j) = \rho.Lp_j, \forall N_j \in N$.*

To calculate the quality of the node, first we need to calculate the pheromone_load. According to **Definition 3.2.2**, we get the initial value of the pheromone_load as $\frac{d}{log(k+1)}$. If we again consider the example of **Definition 3.2.1**, the pheromone_load deposited by the ant on the node N_1 is calculated as $\frac{0.25}{log(1+1)} = 0.830482024$ where the value of d is taken as 0.25 given in **TABLE 3.1**. The evaporation rate (ρ) of the pheromone_load is 0.2, then the quality of the node $(\eta(N_j))$ is computed as $\rho.Lp_j$ which is equal to $0.2 \times 0.830482024 = 0.166096405$.

Definition 3.2.4. *If α be the relative importance of the pheromone_load of the j^{th} node, Lp_j, and β determines the relative importance of heuristic information value or the quality of the node $(\eta(N_j))$, then the probabilities for choosing the next feasible solution component is given by $\boldsymbol{p}(N_j|L^p) = \frac{[Lp_j]^\alpha.[\eta(N_j)]^\beta}{\sum_{N_y \in \mathfrak{N}(L^p)}[Lp_y]^\alpha.[\eta(N_y)]^\beta}, \forall N_j \in \mathfrak{N}(L^p)$.*

Here $\mathbf{p}(N_j|L^p)$ which is the probabilities for choosing the next solution component, is also called the transition probabilities. The value of α and β are given in **TABLE 3.1**. The choice of a solution component $N_j \in \mathfrak{N}(L^p)$ is, at each construction step, done probabilistically on the pheromone_load model. The probability for the N_j is proportional $[Lp_j]^\alpha.[\eta(N_j)]^\beta$.

Theorem 1. *For a given transaction T_i, with deadline $\mathscr{D}(T_i)$, and node N_j in a grid transaction processing system, load balanced transaction scheduling minimizes the load L_i on the node before dispatching the transaction and gives the optimal load, node utilization, throughput, makespan, miss ratio, and load balancing speedup of the grid transaction processing system.*

Proof. Suppose, T and N be the set of the transactions and nodes respectively, where $N_j \in \mathfrak{N}$ and $T_i \in T$. Every transaction has its deadline to complete. If the load of the nodes is known (as explained in **Eq.** (1) and **Eq.** (2)) and the scheduling of the transactions is performed using the optimization approach, the waiting time of the transactions is minimized which enhance the throughput and minimizes the makespan of the transactions. The optimization procedure uses the **Definition** 3.2.1, 3.2.2, 3.2.3, and 3.2.4 to find the optimal nodes. In this procedure, every transaction gets balanced node before scheduling which stops the creation of overhead caused by inter-processor communication. In general, the inter-processor communication is established when a transaction comes to know that its waiting time on a node is increasing and subsequently it searches another node to execute. To avoid the inter-processor communication overhead, the balanced scheduling using optimization approach is an appropriate method. Thus, the transaction scheduling optimization problem formulated in **Eq.** (11) and **Eq.** (12) can be solved. □

The LBTS_ACO is based on ACO (Ant Colony Optimization) approach. In detail, the LBTS_ACO works as follows: When a transaction is submitted to a node in the grid, an ant is initialized, and it starts working. Next step is to find the optimal nodes, the set of the feasible solution, $\mathfrak{N}(L^p)$. At each iteration, exploiting a given transaction_load, solutions to the problem under consideration are constructed probabilistically. Finally, before the next iteration starts, the transaction_load update is performed by using some of the solutions.

Suppose the algorithm starts operation at node N_j. Line 2 finds out the load L_j of the node N_j.

ConstructAntsSolutions: Construction of optimal solution $\mathfrak{N}(\mathfrak{s}^{\mathfrak{p}})$ is the ingredient module of the algorithm. The module assembles the solutions from the finite set of

Algorithm 1 LBTS_ACO

1: start from node N_j ▷ Initial node
2: find L_j
3: make N_j the best-so-far node N_{bs}
4: $k = 1$ ▷ Iteration starts
5: $P_k = 0$
6: **while** $(P_k \leq 1)$ **do**
7: **if** $L_j \leqslant \lfloor \frac{n}{2} \rfloor$ **then**
8: $L_{bs} \leftarrow L_j$
9: $L_j \leftarrow L_j + 1$
10: $N_{bs} \leftarrow N_j$
11: Update the pheromone_load of selected node using **Definition 3.2.2** ▷ pheromone_load update
12: Calculate the quality of the node using **Definition 3.2.3** ▷ Finds quality of the node
13: $L^p \leftarrow L_{bs}$
14: $k = k + 1$
15: $P_k = \frac{log(k)}{log(K)}$
16: **else**
17: select random nodes N_{r1} and N_{r2}
18: find L_{r1} and L_{r2}
19: Calculate $\phi(L_j, L_{r1}, L_{r2})$ Using **Definition 3.2.1**
20: **if** $(0 \leqslant \phi(L_j, L_{r1}, L_{r2}) \leqslant 1.5)$ **then**
21: **if** $L_{r1} \leq L_{r2}$ **then**
22: $L_{bs} \leftarrow L_{r1}$
23: $L_j \leftarrow L_{r1}$
24: $L_j \leftarrow L_j + 1$
25: **else**
26: $L_{bs} \leftarrow L_{r2}$
27: $L_j \leftarrow L_{r2}$
28: $L_j \leftarrow L_j + 1$
29: **end if**
30: **end if**
31: $N_{bs} \leftarrow N_j$
32: Update the pheromone_load of the node using **Definition 3.2.2** ▷ pheromone_load update
33: Calculate the quality of the node using **Definition 3.2.3** ▷ Finds quality of the node
34: $L^p \leftarrow L_{bs}$
35: $k = k + 1$
36: $P_k = \frac{log(k)}{log(K)}$
37: **end if**
38: **end while**
39: $\mathfrak{N}(L^p) \leftarrow \mathfrak{N}(L^p) + L^p$ ▷ Update in the set of feasible solution

solution component N, which starts with an empty partial solution of L^p. The current solution L^p is extended at each construction step by adding a feasible solution component to the set $\mathfrak{N}(L^p)$. A permutation of load distribution subject to the problem constraints is termed as a feasible solution. When the problem constraints are met, the set is determined at each construction step by this solution construction method. This module of the algorithm works as follows: The **while** loop of lines $6-35$ repeatedly selecting the random nodes to search the optimal node for the requested transaction. In each iteration of this **while** loop, the algorithm performs the following operations (see FIGURE 3.1):

Firstly, line 7 checks whether $L_j \leqslant \lfloor \frac{n}{2} \rfloor$. If it is, the following actions are done.

- Line 8 sets L_j to L_{bs}, the best-so-far solution and consequently line 9 increments the load of L_j by 1.
- Line 11 calculates the pheromone_load of N_j using **Definition 3.2.2**.
- Line 11 finds the quality of the node using evaporation rate ρ as elaborated in **Definition 3.2.3**.

If $L_j \nleq \lfloor \frac{n}{2} \rfloor$, the **else** part from line 17 to line 37 searches the nodes until an optimal solution is not found. Thus the algorithm works as follows:

- Line 17 selects the random nodes.
- Line 18 finds the load on those selected random nodes as we see in Eq. 1 and Eq. 3.
- Line 19 calculates $\phi(L_j, L_{r1}, L_{r2})$ as depicted in **Definition 3.2.1**.
- Line 20 checks the value of $\phi(L_j, L_{r1}, L_{r2})$ whether it is within $[0, 1.5]$. If it is, then line 21 again checks the load between N_{r2} and N_{r1} and one that is less loaded is set to the L_{bs}. Here the node which has less load is preferred first and is assigned as the best-so-far node. Otherwise, lines 26-28 select another node L_{r2} and do the same for L_{r2} as lines 22-24 do for L_{r1}.

We repeat the iterations of the **while** until all the transactions are not scheduled.

UpdatePheromones: Line 32 calculates the pheromone_load value on the node N_j deposited by the ants as depicted in **Definition 3.2.2**. Line 33 calculates the quality of the best-so-far node N_{bs} using the value of evaporation rate ρ which is 0.2 as described in **Definition 3.2.3**. Then Line 34 sets the best-so-far solution to the L^p.

DaemonActions: Line 39 updates the set of feasible solution $\mathfrak{N}(L^p)$.

FIGURE 3.1 shows the working example of the LBTS_ACO.

3.2.2 Applying the LBTS_ACO Algorithm

The characteristics of balanced scheduling in the grid transaction processing systems are different from those of the other scheduling problems that have been solved by ACO in the aspect that the solution to balanced scheduling problem is an unordered subset of balanced nodes. The LBTS_ACO conducts to explore the power set of the set of balanced nodes. In our study model, the grid transactions are represented by a complete undirected graph. Let $\mathscr{G} = (\mathfrak{C}, \mathscr{E})$ denotes the complete undirected graph representing the grid transactions, where $\mathfrak{C}$ is the set of nodes; $\mathscr{E} = \mathfrak{C} \times \mathfrak{C}$ is the set of edges between the nodes. **FIGURE 3.1** gives an illustrative example how the LBTS_ACO works. Suppose there are m number of the transactions $(T_1, T_2, ..., T_m)$ which arrive at the system with available nodes (suppose $N = 8$).

ConstructAntsSolutions: The LBTS_ACO constructs a set of the feasible solution so that the scheduler gets the optimal nodes for scheduling the incoming transactions. Construction of the solution starts from the initial node N_1. As **FIGURE 3.1**(a) illustrates, the LBTS_ACO works by checking the load of the node N_1. Since $L_1 \leqslant \lfloor \frac{n}{2} \rfloor$ (since $n = 8$), the condition becomes true. Thus, the N_1 is selected as the best-so-far solution. The load value is incremented by 1, and it becomes 4. Then the ant in the algorithm comes out of the **while** loop. The last position of the ant is still at the node N_1. Therefore, again line 5 checks the condition. This time also, node N_1 is selected, and the load is incremented to 5. Now, the condition becomes false when checked, so two random nodes N_2 and N_6 are selected as shown in **FIGURE 3.1**(b). Their loads are 8 and 6 respectively. Then the algorithm finds the value of δ_{126} and δ_{162} as 3.0 and 0.33 respectively. Then $\phi(L_1, L_2, L_6)$ is calculated by selecting $\mathbf{min}(3.0, 0.33)$. Here $\phi(L_1, L_2, L_6) \leqslant 1.5$, hence the algorithm selects the node N_6 which has the minimum load between them. The load value of N_6 is incremented by 1 and it becomes 7. In subfigure (c), the ant is at N_6. Line 5 checks the condition. Since it is false, two random nodes N_3 and N_5 with their respective load 5 and 4 are selected. δ_{635} and δ_{653} are calculated as 0.66 and 1.5 and the $\mathbf{min}(0.66, 1.5)$ value is selected. Here N_5 is selected as the solution. The same procedure continues in **FIGURE 3.1**(c) to **FIGURE 3.1**(f). The solution is kept in the scheduling queue.

UpdatePheromones: After selection of the best-so-far solution, the quality of the selected node is calculated, and the ant trails the pheromone_load on the selected node.

In subfigure (a), when node N_1 is selected the first time, then pheromone_load value which is dependent on the initial pheromone_load and the number of iteration value, i.e., $\frac{d}{log(k+1)}$ (as depicted in **Definition 3.2.2**) is calculated as $\frac{0.25}{log(1+1)} = 0.830482024$. The quality of the node $\eta(N_1)$ (as depicted in **Definition 3.2.3**) is calculated as $\rho * Lp_1$ where ρ is 0.2 and $Lp_1 = 0.830482024$. Therefore, $\eta(N_1)$ is calculated as 0.166096405. The ant is still at the node N_1 because $L_1 \leqslant \lfloor \frac{8}{2} \rfloor$. Now the pheromone_load L_1 of N_1 is updated as $(1-0.2) * 0.830482024 + \frac{0.2}{2} * 0.830482024$ which is 0.747433822. This value is deposited at iteration $k = 2$. As the iteration will keep on increasing; the pheromone_load will keep on decreasing with evaporation rate $\rho = 0.2$.

At iteration $k = 3$, in subfigure (b), when L_1 becomes 5, the ant at N_1 compares loads of two randomly selected nodes N_2 and N_6. After selecting N_6 as the best-so-far solution, the pheromone_load L_6 is calculated by $(1.0 - 0.2) * 0.747433822 + (0.2/3) * (0.830482024 + 0.747433822)$ which gives 0.703141447.

The quality of the selected node $\eta(N_6)$ is calculated as $\rho * Lp_j$ which is 0.140628289. We observe that the quality of the node gets decreasing with the load increase on the node.

DaemonActions: This module of the LBTS_ACO updates the set of feasible solution $\mathfrak{N}(L^p)$ globally by generating a scheduling queue as shown in **FIGURE 3.1**(f).

The meta-heuristic algorithms like ACO suffer from premature convergence [108] when applied for scheduling problem, we deal this problem by using following approaches in our algorithm.

3.2.3 Prevention of Premature Convergence of the Algorithm

We incorporate the parameter P_k in the algorithm to achieve the prevention of premature convergence in the LBTS_ACO algorithm as:

$$P_k = \frac{log(k)}{log(K)} \tag{13}$$

where k is the counter for the number of iterations and K is the maximum number of iterations. Here P_k represents the probability of avoiding newer solutions where $0 \leq P_k \leq 1$. Each time P_k is compared to a randomly generated quantity P_{event}.

When k value increases, the probability of the event $P_{event} > P_k$ decreases (suppose P_{event} is a randomly generated number which lies in the range $[0,1]$) i.e., at the lower value of k, the probability of searching new nodes by the ants is higher and at higher values of k, the probability of new search decreases. Thus, the algorithm can prevent a very quick convergence to locally optimized solution.

3.2.4 Stagnation Avoidance

Another undesirable situation, i.e., stagnation [108] may arise when all ants construct the same solution over and over again. This situation prevents the generation of new search. It happens when the parameters (α, β, ρ) of the LBTS_ACO algorithm are not well tuned for taking the problem. If the value of ρ is too high, the stagnation situation may take place. Therefore, we have set the values of the parameters as $\alpha = 0.5$, $\beta = 0.5$, and $\rho = 0.2$ as given in **TABLE 3.1**.

3.2.5 Convergence Test

The convergence of the LBTS_ACO algorithm is the first theoretical problem which means if the proposed algorithm can find the optimal solution when given enough resources. As the proposed algorithm is a stochastic search procedure, the pheromone update may prevent it to even reach an optimum. Typically, there are at least two types of convergence of the proposed algorithm which can be considered [54]: ***convergence in value*** and ***convergence in solution***.

Convergence in value also known as **Asymptotic convergence** evaluates the probability that the algorithm generates an optimal solution at least once. ***Convergence in solution*** known as **Reachability convergence** evaluates the probability that the algorithm reaches a state which keeps on generating the same optimal solution.

Proposition 2. *Given **Algorithm 1** that using the pheromone_load update rule from Definition 3.2.3 for any pheromone_load value, the following holds*

$$\lim_{k \to \infty} L_{p_j}(k) \leq \frac{Lp_j.K}{\rho} \tag{14}$$

where $L_{p_j}(k)$ denotes the pheromone value L_{p_j} at iteration k while K is the maximum iteration.

Proof. At any iteration, the maximum possible increase of pheromone_load value L_{p_j} is $\eta(N_j)$ if all solution are equal to N_j with new choice of solution. Therefore, due to evaporation, the pheromone_load L_{p_j} at iteration k is bounded by

$$Lp_j \leftarrow (1-\rho)^k . \frac{d}{log(k+1)} + K . \sum_{k=1}^{K} (1-\rho)^{K-k} . Lp_j \tag{15}$$

where $\frac{d}{log(k+1)}$ with d being constant is the initial value of all the pheromone_load trail parameters. Asymptotically, because $0 < \rho \leq 1$, this sum converges to $\frac{Lp_j . K}{\rho}$. □

From this proposition, we can say that the pheromone value upper bound in the pheromone update rule is $\frac{Lp_j . K}{\rho}$.

Theorem 3. *Let $P_s(k)$ is the probability that an algorithm generates an optimal solution in the k^{th} iteration, then the algorithm has asymptotic convergence and reachability convergence if $\lim_{k\to\infty} P_s(k) = 1$.*

Proof. From Proposition 1, we get that minimum value of pheromone_load is greater than 0, because it is anyway bounded by maximum pheromone_load value. Since minimum pheromone_load > 0, at each iteration, any generic solution can be generated with a probability greater than 0. Therefore, the probability of generating an optimal solution tends to 1 even at a sufficiently large number of iterations. Therefore, we state that the algorithm is guaranteed to find an optimal solution with a probability that can be made arbitrarily close to 1 if given enough time (convergence in value).

□

3.3 Simulation and Results Analysis

The proposed scheduling algorithm, LBTS_ACO, is evaluated through simulations with Colored Petri Nets (CPNs or CP-nets). The transactions are generated randomly using

exponential distribution. For modeling failures in the system, we use the Poisson process. In our simulations, the system scenario is based on Czech National Grid Infrastructure Metacentrum project. In our simulations, the grid scenario is based on Czech National Grid Infrastructure Metacentrum project. The transaction traces used in the simulations specify a set of parameters such as the transaction identifier, associated transaction user priority, the set of properties to be met in the target resource and arrival time to the scheduler. Some assumptions are made for the simulation. These are:

- The workload consists of sets of independent transactions on a heterogeneous grid system.
- A maximum deadline for the transactions is 1000 time units.

We compare the performances of the proposed algorithm with five existing scheduling algorithms in the literature namely, Extremal Optimization [2, 27, 28], GA [33, 1], Hierarchical Load Balanced Algorithm (HLBA) [34], Dynamic and Decentralized Load Balancing (DLB) algorithm [35], and Randomized algorithm with random selection method [36]. For the purpose of comparison, we simulated all the scheduling algorithms also on the non-transaction processing system.

The short introduction about each of the mentioned algorithms is presented below.

- Extremal Optimization: Extremal Optimization (EO) is a nature-inspired optimization technique. This technique has moderate computational complexity and small memory requirements. It is a meta-heuristic approach.
- Genetic Algorithm: In the GA, the candidate solutions (called individuals) and their abstract representations (named chromosomes), are improved in each iteration to finally get the optimum solution. It uses selection operation to find the survival for each individual. Thus, the fitness of the whole population is determined. Based on the fitness value, the individuals are selected randomly from the population. The individuals that have high fitness value are inherited in the next generation with a higher probability while the individuals with low fitness are inherited in the next generation with a smaller probability.

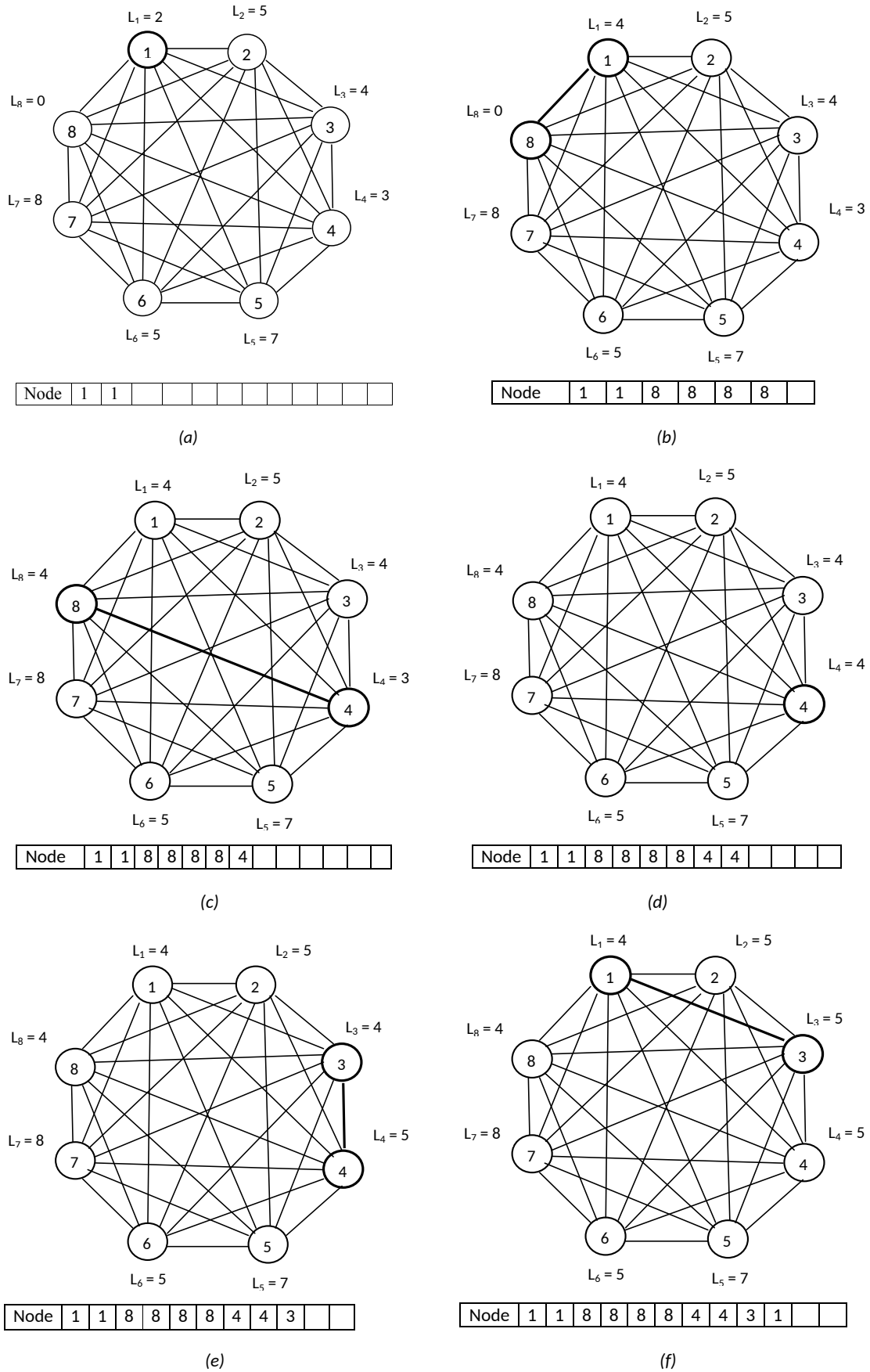

FIGURE 3.1: Working example of the LBTS_ACO when N=8

- HLBA: In this algorithm, the system load is used as a parameter in determining a balance threshold. And the scheduler adapts the balance threshold dynamically when the system load changes. The scheduling algorithm balances the system load with an adaptive threshold, and it minimizes the makespan of jobs. This algorithm is suitable for a dynamic grid environment, but in the case of the grid transaction processing the resulting schedule is not optimal.

- DLB: This algorithm is for computationally intensive jobs on a heterogeneous distributed computing platform. The time spent by a job in the system is considered as the main issue that needed to be minimized. The algorithm uses site desirability for processing power and transfer delay to guide load assignment and redistribution. The communication overhead involved in the information collection is reduced using mutual information feedback. The algorithm focuses on the communication overhead involved in capturing load information of sites before making a dispatching decision. For the data grids like the grid transaction processing, the algorithm is not able to provide good results.

- Randomized: In this algorithm, the nodes are selected randomly using exponential distribution. This algorithm uses the first-in-first-out approach.

All the experiments are carried out in two different ways; first in the grid transaction processing system i.e., using Transaction Management (TM) and second in the traditional grid processing system without Transaction Management (WTM). The final results are produced on an average basis.

We addressed the parameters optimization analysis together with the convergence behavior in section 3.2.3, section 3.2.4 and section 3.2.5. We also conduct statistical tests to further analyze the validity of results. For the best final result, the normality of data with Shapiro-Wilks test is studied. **TABLE 3.2** shows the confidence value (p-value). As the p-value ≤ 0.5, the null hypothesis that the samples came from normal distribution must be rejected. Similarly, for the another objective function, i.e., makespan, **TABLE 3.3** shows that the null hypothesis of the samples being in normal distribution must be rejected.

We have also conducted nonparametric tests (see **TABLE 3.4**) to check the difference among the methods using Wilcoxon or Mann-Whitney test [109]. The observations in

TABLE 3.4 shows that p-value ≤ 0.5. Therefore, it can be concluded that the LBTS_ACO outperforms all other algorithms. Therefore, the simulations in **TABLE 3.5** show that the LBTS_ACO is faster than EO, GA, HLBA, DLB, and Randomized algorithms.

TABLE 3.2: Normality Shapiro-Wilk tests for the best results of load

Algorithm	Shapiro-Wik W	p-value
LBTS_ACO	0.889405172	0.0022
EO	0.889830107	0.0024
GA	0.88694925832	0.00258
HLBA	0.88827583596	0.00258
DLB	0.89029488641	0.00258
Randomized	0.89111842344	0.00265

TABLE 3.3: Normality Shapiro-Wilk tests for the best results of makespan

Algorithm	Shapiro-Wik W	p-value
LBTS_ACO	0.8668013739216988	< 0.001
EO	0.8642371443799678	< 0.001
GA	0.8558035724664631	< 0.001
HLBA	0.9207793391610051	0.017
DLB	0.7852026082946301	< 0.001
Randomized	0.9328940495822504	0.043

TABLE 3.4: Wilcoxon statistical tests for the best results (load) found for LBTS_ACO, EO, HLBA, GA, DLB, and Randomized algorithms. Assume null hypothesis $\mu_0 = 0$ and null hypothesis: two-sided, $\hat{\mu} < \mu$

Observation	Wilcoxon	p-value	95% Confidence Interval	$\hat{\mu}$
LBTS_ACO vs. EO	720	0.351641	$-\infty$ 19.1000	-1.09331184504
LBTS_ACO vs. GA	730	0.24793775	$-\infty$ 17.5000	-1.15169113147
LBTS_ACO vs. HLBA	728	0.245388	$-\infty$ 16.6000	-2.15169113147
LBTS_ACO vs. DLB	576	0.015653	$-\infty$ -4.5000	-36.8436479761
LBTS_ACO vs. Randomized	480	0.001042	∞ -29.0000	-70.9136724848

3.3.1 Load on the System

This experiment compares the load of each method independently. **TABLE 3.6** illustrates the result of load measurement. In **FIGURE 3.2** we see the load on the grid transaction processing system for all these algorithms. As expected, our proposed algorithm outperforms the others. When comparing the results of the algorithms, it can be observed that the gap between these curves widens as the completion time increases. However, these curves are almost the same when the completion time is less than 800 time unit.

TABLE 3.5: Load on the system with 40 simulations

Strategy	Iteration	Average	Standard deviation	Confidence Interval (95%)	Min
LBTS_ACO	300	21.1794871795	5.34392439525	9.5 28	9
	200	16.07	5.31753082946	4.5 27	4
	100	8.81	5.91868087686	6.25 21	6
EO	300	22.454	6.4430243	11.5 38	11
	200	18.05	6.3075946	11.5 35	11
	100	9.5	6.05675	11.5 30	11.5
GA	300	23.75656	6.545430987	13.5 45	13
	200	21.575	6.35657525	13.0 45	13
	100	20.3567	6.0567654	13.0 38	13
HLBA	300	28.575	7.87567834	21.55 55	21
	200	27.955	7.35657525	21.55 52	21
	100	26.575	7.20567654	21 50	21
DLB	300	35.45	8.457237	25.55 60	25
	200	35.05	8.35657525	25.5 58	25
	100	34.95	8.0567654	25.05 55	25
Randomized	300	44.89	10.576565	25.78 75	25
	200	44.50	10.35657525	25.50 72	25
	100	44.257	10.0567654	25.5 70	25

TABLE 3.6: Load on the system

Time (time unit)	LBTS_ACO		EO		GA		HLBA		DLB		Randomized	
	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM
100	5.8	7.2	6	7.5	6	6.2	6.2	8	8	9	8	5.8
200	15.8	17.8	16.1	18.5	16.5	16.75	16.9	19.5	18	21.5	23.8	24
300	28	30.8	28.5	31.5	29.05	29.55	30.2	35.7	35.3	45.5	44.8	47.4
400	43.5	48.5	44.5	45.5	44.75	48.5	45.78	51.5	58.8	61.5	94.4	70.33
500	60.6	57.6	62.5	58.75	62.5	65	62.7	65.6	100	69.7	137	81.2
600	63.6	51.2	63.75	55	64.5	59.5	65.7	75.7	125.5	79.9	155.2	105.2
700	85	64.4	86.6	65.5	85.0	78.5	85	79.5	150.9	100.5	157.8	122.66
800	125.4	72.4	126.5	75.5	128.5	80.25	129.6	81.3	200.5	125.6	217.6	150.6
900	146.4	77.4	148.25	79.75	150.75	83.75	152.4	85.4	225.6	181.5	259.8	195.8
1000	169	73.2	170.75	79	173.5	85	181	90	251	180	265	201.7

For the long-lived transactions, the system load becomes heavier, and thus the ACO has more chance to manage the load, thereby making the load balanced scheduling decisions and bringing in better performance. This shows that the LBTS_ACO works well, especially in higher load scenarios and for the long-lived transactions. On the other hand, the DLB and the Randomized algorithms underperform in the comparison of the LBTS_ACO, EO, GA and HLBA, because they schedule the transactions to random nodes. The randomly selected nodes may be either lightly loaded or heavily loaded. Therefore they waste some processing and switching time when they encounter heavily loaded nodes.

Comparison of the load on the grid processing system without transaction management for all the algorithms are shown in **FIGURE 3.3**. In this case also, the proposed algorithm

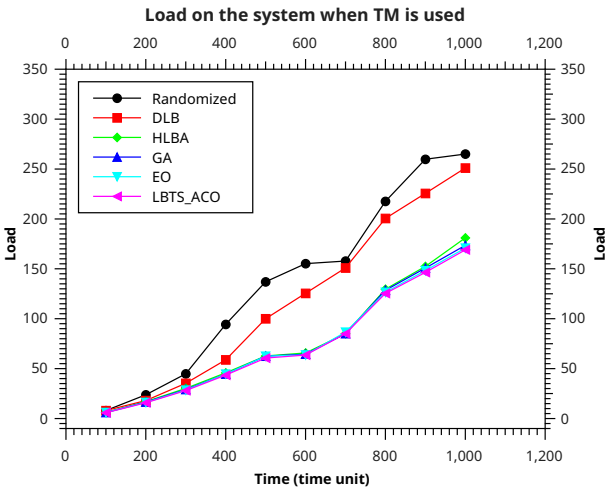

FIGURE 3.2: Load on the system with Transaction Processing

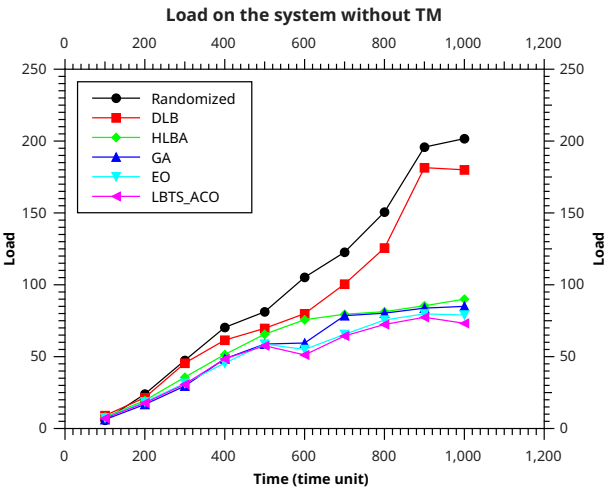

FIGURE 3.3: Load on the system without Transaction Processing

outperforms the other mentioned algorithms. As the completion time increases, the gap among all the curves widens. When comparing the results of the LBTS_ACO with others, the widening gap among their curve prove that the LBTS_ACO works better than others. The LBTS_ACO performs better than others, not only for the long-lived transactions but also for the short-lived transactions.

In **FIGURE 3.4** we have the comparison of results of the LBTS_ACO between the grid transaction processing system and traditional grid processing system. This shows that the LBTS_ACO is a better approach for balanced scheduling not only for the grid transaction processing system but also for traditional grid processing system. The gap among the curves widens hugely as the completion time increases further from 500 time unit. It is observed that the LBTS_ACO manages the load better in traditional grid processing system than how it manages the load in the grid transaction processing system. The reason behind the huge gap between the two curves is that a transaction roll back all its sub-transactions when one of the sub-transactions is rolled back and again transaction request is resumed if the deadline has not been missed.

In **FIGURE 3.5** we see the load deviation comparison from iterations 100 to 300 using Eq.5. It is observed that the load deviation in LBTS_ACO is lesser at the increased iterations than other algorithms. It means the LBTS_ACO performs better than others in load balancing.

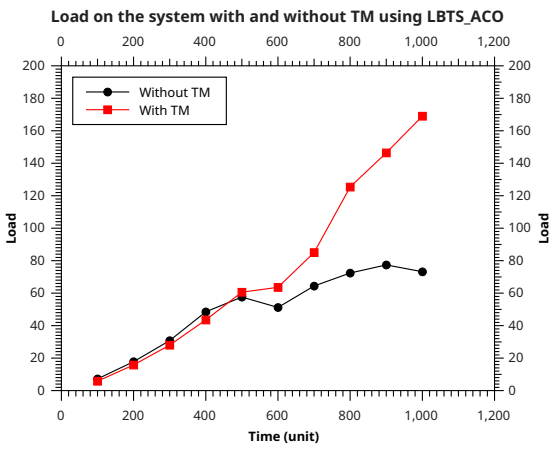

FIGURE 3.4: Load on the system with and without Transaction Processing

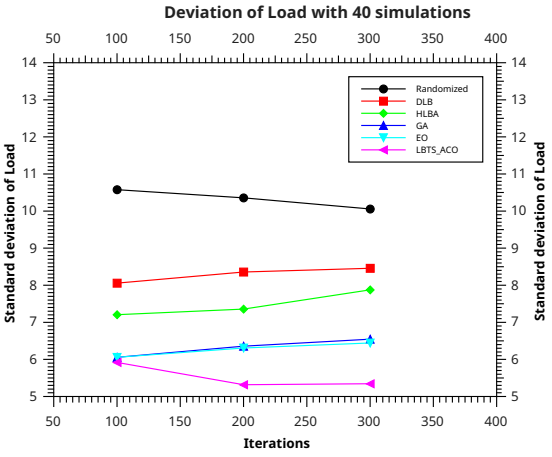

FIGURE 3.5: Load Deviation results

3.3.2 Node Utilization

This experiment compares the proposed algorithm with the mentioned algorithms regarding the node utilization. The node utilization indicates how well the loads are balanced across all the nodes involved in a grid processing system. The higher node utilization will ensure the better performance of load balancing. In **TABLE 3.7** we have the comparison of results among the mentioned algorithms.

TABLE 3.7: Node Utilization

Time (time unit)	LBTS_ACO		EO		GA		HLBA		DLB		Randomized	
	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM
100	0.924	0.911	0.905	0.9	0.8575	0.825	0.644	0.667	0	0	0	0
200	0.95	0.979	0.925	0.95	0.9125	0.925	0.924	0.911	0	0	0.5	0.5
300	0.949	0.972	0.935	0.952	0.925	0.945	0.943	0.946	0.865	0.8	0.834	0.778
400	0.966	0.977	0.955	0.955	0.945	0.957	0.967	0.956	0.9095	0.8642	0.889	0.834
500	0.9752	0.99	0.9725	0.98	0.9715	0.9798	0.976	0.971	0.9361	0.908	0.92	0.886
600	0.9849	0.987	0.9825	0.9865	0.9825	0.98	0.979	0.9783	0.9549	0.949	0.944	0.939
700	0.9846	0.984	0.9825	0.983	0.981	0.9825	0.979	0.9819	0.9606	0.9649	0.951	0.956
800	0.9883	0.985	0.9875	0.9835	0.9835	0.9855	0.986	0.9832	0.9589	0.9697	0.949	0.962
900	0.9929	0.99	0.9925	0.9898	0.991	0.9875	0.984	0.9876	0.9704	0.9746	0.944	0.968
1000	0.996	0.989	0.995	0.9875	0.993	0.9865	0.9893	0.9876	0.977	0.9784	0.971	0.973

Depicted in **FIGURE 3.6**, it can be seen that the node utilization in the grid transaction processing system for the proposed algorithm is higher than other algorithms. The LBTS_ACO, EO, GA and the HLBA are approximately giving better results but the LBTS_ACO outperforms others when the transactions are short-lived (having deadlines less than 200 time unit). The reason is that the LBTS_ACO dominates the load balancing

process when the system workload is light. The other reason is that the LBTS_ACO also manages the overhead caused by the interprocess communication.

In **FIGURE 3.7** we have the comparison of the node utilization in grid processing system without the transaction management among the mentioned algorithms. Once again the LBTS_ACO outperforms the other algorithms. When comparing the results of the LBTS_ACO, EO, GA, and the HLBA, it is observed that the gap among their curves remains widened when the completion time is less than 700 time unit.

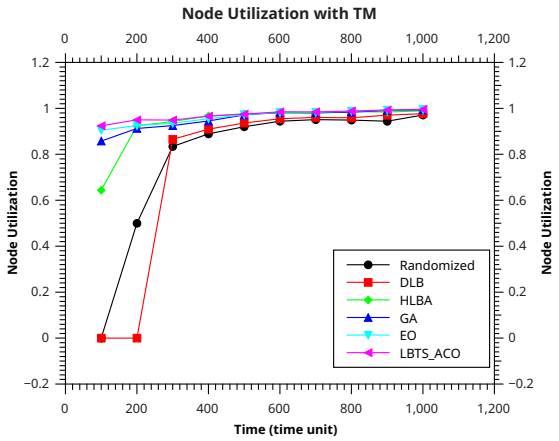

FIGURE 3.6: Node Utilization with Transaction Processing

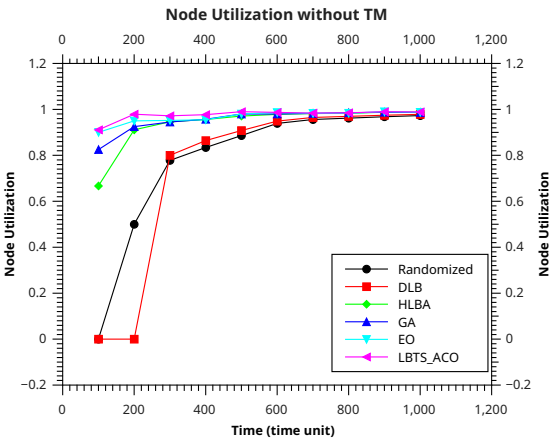

FIGURE 3.7: Node Utilization without Transaction Processing

In **FIGURE 3.8** we have the comparison of the node utilization of the LBTS_ACO between the grid transaction processing system and grid processing system. It is observed that for the short-lived jobs in the computational grids and the short-lived transactions in the data grids (up to 600 time units), the node utilization in the computational grids is much higher than that in the data grids. This huge difference in node utilization is caused due to the deadline constraints in the data grids. But in the case of the long-lived jobs in the computational grids and the long-lived transactions in the data grids, the difference of the node utilization between them is opposite. Thus, the LBTS_ACO algorithm gives better result not only for the data grids (with TM) but also for the computational grids (WTM).

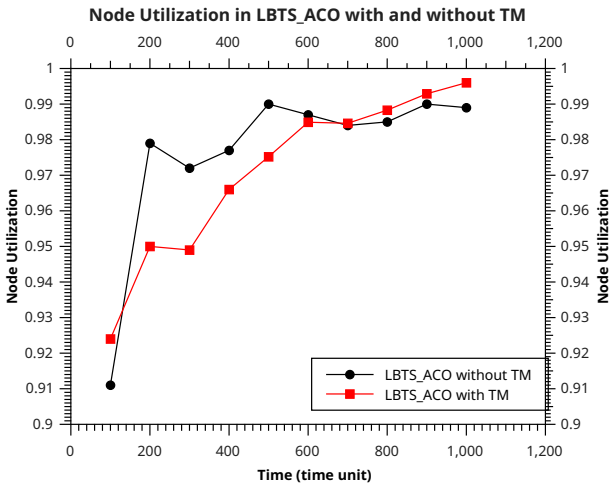

FIGURE 3.8: Node Utilization with and without Transaction Processing

3.3.3 Transaction Throughput

This experiment evaluates the performance among the algorithms concerning throughput. Here the average throughput of the grid transaction at different processing levels are presented. **TABLE 3.8** illustrates the result of throughput measure. The transaction throughput which has been defined in **Eq.** (9) is first measured for the grid transaction processing system (as depicted in **FIGURE 3.9**), and then measured in traditional grid processing system (as depicted in **FIGURE 3.10**). **FIGURE 3.9** and **FIGURE 3.10** show that the LBTS_ACO algorithm outperforms the EO, GA, Randomized, DLB, and HLBA algorithms.

TABLE 3.8: Throughput (Number of the transactions per unit time) after executing for a particular time unit

Time	LBTS_ACO		EO		GA		HLBA		DLB		Randomized	
	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM
100	0.01	0.016	0.01	0.015	0.01	0.0156	0.01806	0.02	0	0	0	0
200	0.094	0.135	0.09	0.125	0.098	0.0176	0.12	0.1316	0	0	0	0
300	0.134	0.29	0.124	0.275	0.125	0.27	0.1643	0.203	0.0254	0.02966	0.02	0.02333
400	0.164	0.4905	0.154	0.4565	0.152	0.425	0.2174	0.3193	0.03818	0.0477	0.03	0.0375
500	0.3616	0.6692	0.3565	0.5656	0.3457	0.56	0.39018	0.40464	0.0585	0.0789	0.046	0.062
600	0.5623	0.755	0.5575	0.725	0.545	0.715	0.4817	0.5247	0.176	0.193	0.1383	0.15166
700	0.8748	0.7718	0.87	0.77	0.85	0.77	0.64035	0.77455	0.2781	0.2745	0.2185	0.2157
800	1.05	0.827	1.05	0.815	0.95	0.81	0.84386	0.80387	0.2911	0.31976	0.22875	0.25125
900	1.9193	0.9477	1.9	0.925	0.8585	0.915	0.81027	0.971	0.4002	0.42565	0.3144	0.3344
1000	3.198	1.0106	2.5	0.99	0.99	0.9879	1.01419	0.9517	0.4989	0.54472	0.392	0.428

In **FIGURE 3.9** we see that for the short-lived transactions with the completion time less than 500 time units, the LBTS_ACO and the EO, GA, and HLBA perform almost the

same. But if the completion time is more than 500 time units, the LBTS_ACO outperforms the HLBA algorithm and after 800 time units it outperforms all the algorithms. That means the chances of deadline miss in the long-lived transactions becomes considerably much less in the LBTS_ACO than others. The reason is that the load balanced scheduling performed by the proposed algorithm reduces the waiting time on the nodes. Thus, the chances of getting the resources increases which enhance the throughput of the system. The other benefit for the long-lived transactions is that they have longer deadlines which control transaction abort.

While in **FIGURE 3.10** we see that in the computational grids (jobs without TM), the LBTS_ACO always performs better than other algorithms. When comparing the LBTS_ACO with others, it is observed that for the short-lived jobs (less than 700 time units), the gap among their curves widens. The widened curves show that the LBTS_ACO works much better than others. For the long-lived jobs, the LBTS_ACO and others perform almost similar. The reason is that the load balancing process in the LBTS_ACO is dominated in the comparison of other algorithms and thus it has more chance to make balanced scheduling decisions. The number of successful tasks' execution increases and thus the throughput of the grid processing system increases.

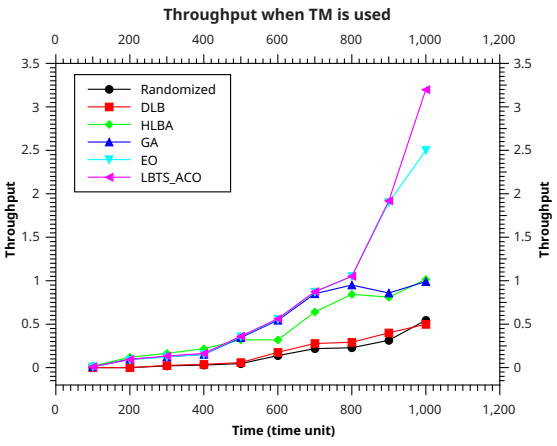

FIGURE 3.9: Throughput when TM is used is used

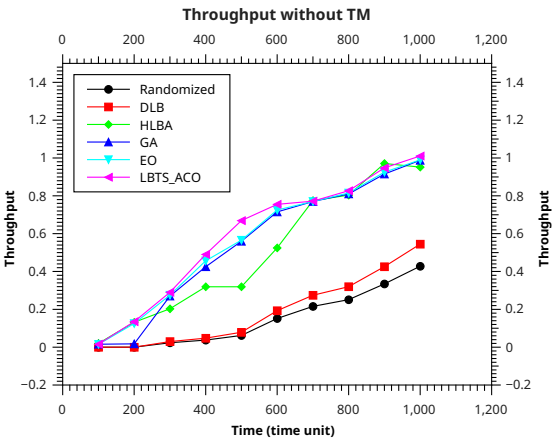

FIGURE 3.10: Throughput when no TM is used

The results also show that the throughput of the LBTS_ACO in the grid transaction processing system (with TM) is better than its throughput in grid processing system (WTM). **FIGURE 3.11** shows that for the short-lived transactions and jobs, the LBTS_ACO gives better throughput result for the computational grid than the data grid.

But for long-lived transactions and jobs, the LBTS_ACO gives better throughput result for the data grid than the computational grid. There is a huge gap between the two curves after 700 time unit. The reason is that in the grid transaction processing system the transactions always have deadlines within which they are forced to run completely. For the long-lived transactions, if the load is light, the waiting time will be less, thereby executing more number of transactions within their deadlines.

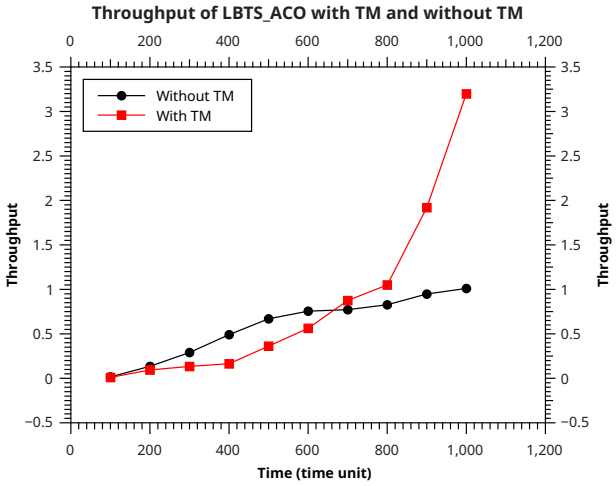

FIGURE 3.11: Comparison of the LBTS_ACO Throughput

3.3.4 Makespan Analysis

In this experiment, the makespan of the mentioned algorithms are evaluated and are compared. Since there may be several transactions in the grid, the simulations are done on the size and the number of the transactions. In such dynamic situation, how these algorithms respond is discussed here. **TABLE 3.9** shows the makespan along several iterations, i.e., 100, 200, and 300 in 40 simulations. It presents the mean result achieved by the populations with the associated standard deviation and 95% confidence interval and the best result (Max). In **TABLE 4.21**, we choose from 100 to 1000 transactions and compare their makespan as illustrated in **TABLE 4.21**. The table shows that the makespan taken for all algorithms grows up as the number of the transactions or tasks increases.

TABLE 3.9: Makespan with 40 simulations

Strategy	Iteration	Average	Standard deviation	Confidence Interval (95%)	Max
LBTS_ACO	300	115.225	24.075389405	60.5 160	160
	200	96.875	21.9933938945	50.5 143	143
	100	71.425	23.7106384351	25.25 130	130
EO	300	122.454	26.4430243	61.5 180	180
	200	120.5	26.3075946	60.5 175	175
	100	119.5	25.505675	60 175	175
GA	300	130.75656	27.545430987	73.5 205	205
	200	128.575	26.935657525	73.0 195	195
	100	128.3567	26.0567654	73.0 195	195
HLBA	300	135.75	27.87567834	81.55 215	205
	200	135.5	27.35657525	81.55 210	210
	100	134.575	27.20567654	80.5 200	200
DLB	300	180.75	28.457237	150 350	350
	200	180.05	28.35657525	150 348	348
	100	178.75	28.0567654	150 345	345
Randomized	300	230.89	35.567894	190 380	380
	200	230.50	35.35657525	190 375	375
	100	230.25	35.0567654	190 370	370

TABLE 3.10: Makespan

Transactions	LBTS_ACO		EO		GA		HLBA		DLB		Randomized	
	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM
100	310	315	313	320	325	320	328.6	351.85	530.35	505.84	675	643.8
200	375	353	419	354	420	354	421.6	387.5	627.78	607.51	799	773.2
300	424	412	425	415	430	417	531.65	461.125	695.357	677.91	885	862.8
400	530	538	535	541	540	541	542.5	542.5	792	745.171	1,008	948.4
500	570	560	575	565	580	570	581.25	577.375	853.285	829.321	1,086	1,055.5
600	575	572	585	583	600	588	608.375	589	908.757	901.214	1,156.6	1,147
700	635	640	640	650	664	665	674.25	675.8	922.428	985.914	1,174	1,254.8
800	700	705	705	735	725	740	730.05	744.375	982.142	1,039.65	1,250	1,323.2
900	720	735	715	755	700	765	763.375	767.25	1,012.157	1,075.8	1,288.2	1,369.2
1000	720	735	725	775	735	805	799.8	819.125	1,131.428	1,397.94	1,440	1,779.2

We observed from **FIGURE 3.12** that the makespan of the LBTS_ACO, EO, GA and HLBA are very close and are much better than the Randomized and the DLB algorithms. The remarkable point in this experiment is that the LBTS_ACO gives reduced makespan for the batch of a large number of the transactions. This becomes possible due to the balanced scheduling which reduces the waiting time on the nodes resulting more number of the transaction commit.

When we compare these algorithms by simulating in the computational grid processing systems (WTM) with respect of makespan, we observed that the LBTS_ACO, EO, GA and the HLBA outperform other two algorithms. In the comparison of the EO, GA, HLBA, DLB, and Randomized, the LBTS_ACO performs better (as shown in **FIGURE 3.13**).

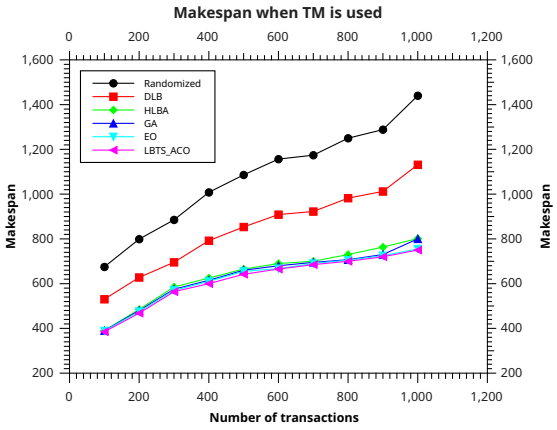

FIGURE 3.12: Makespan when TM is used

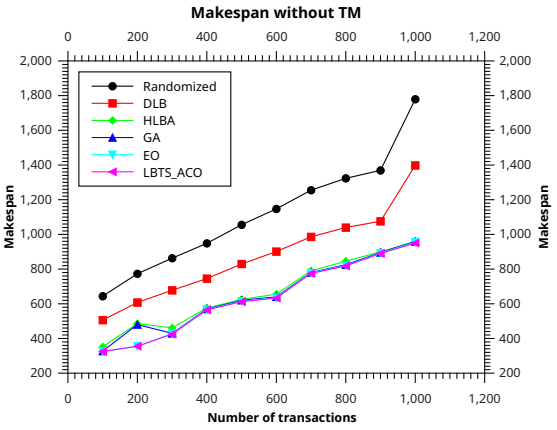

FIGURE 3.13: Makespan when no TM is used

FIGURE 3.14 illustrates the comparison of makespan of the LBTS_ACO in the data grid (with TM) and that in the computational grid (without TM). If the number of tasks in a batch is less than 600, the LBTS_ACO takes less time for the computational grid than what it takes for the same number of the transactions in the data grid.

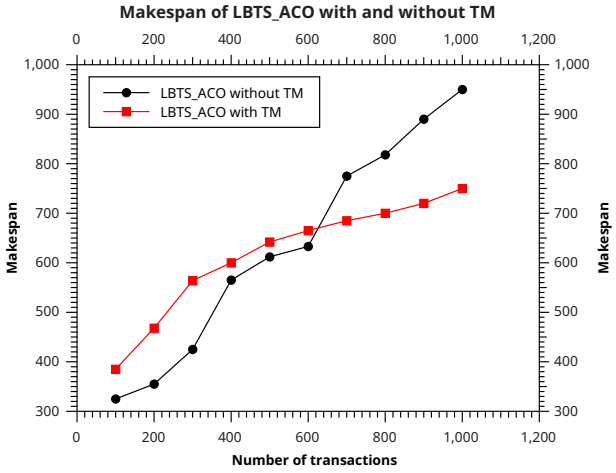

FIGURE 3.14: Comparison of the LBTS_ACO makespan

3.3.5 Miss Ratio

In this experiment, miss ratio is compared among the mentioned algorithms. The comparison results are shown in **TABLE 3.11**.

TABLE 3.11: Miss Ratio (%)

Time (time unit)	LBTS_ACO		EO		GA		HLBA		DLB		Randomized	
	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM
100	0	0	0	0	0	0	35.6	33.3	0	0	0	0
200	5	2.1	6.5	3.5	6.75	3.65	7.6	8.9	0	0	50	50
300	5.1	2.8	5.5	3.1	5.65	3.5	5.7	5.4	13.5	20	16.6	22.2
400	3.4	2.3	3.5	2.35	3.56	2.45	3.3	4.4	9.05	13.58	11.1	16.6
500	2.4	1.0	2.45	1.5	2.5	1.55	2.4	2.9	6.39	9.20	8.0	11.4
600	1.51	1.3	1.525	1.35	1.67	1.45	2.1	2.17	4.51	5.1	5.6	6.1
700	1.54	1.6	1.6	1.75	1.65	1.8	2.1	1.81	3.94	3.51	4.9	4.4
800	1.17	1.5	1.2	1.575	1.25	1.6	1.4	1.68	4.11	3.03	5.1	3.8
900	0.72	1.0	0.75	1.25	0.85	1.3	1.6	1.24	2.96	2.54	5.6	3.2
1000	0.4	1.1	0.575	1.25	0.75	1.3	1.07	1.24	2.3	2.155	2.9	2.7

We see in **FIGURE 3.15** which compares the miss ratio of the proposed algorithm with that of the mentioned algorithms. It can be seen that the LBTS_ACO has the minimum miss ratio as compared to others. The transaction whose completion time is more than 300 time unit, the miss ratio of the LBTS_ACO, EO, GA and HLBA are approximately same. However, the LBTS_ACO performs better than others. The reason is that all transactions tend to compete for resources and due to the load balanced scheduling they readily get the chance to obtain their resources, thereby reducing the miss ratio.

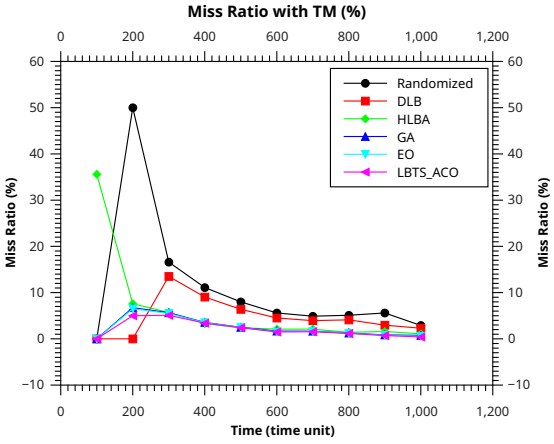

FIGURE 3.15: Miss Ratio with Transaction Processing

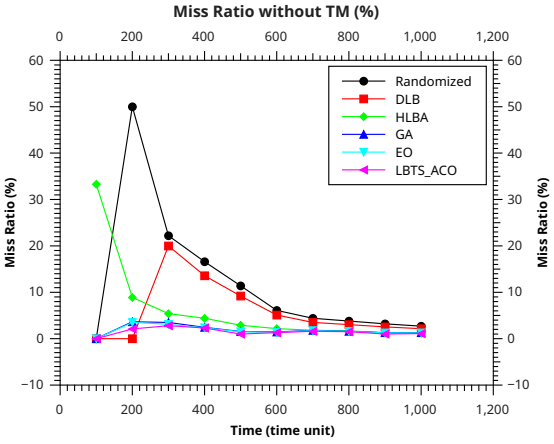

FIGURE 3.16: Miss Ratio without Transaction Processing

We see in **FIGURE 3.17** which compares the miss ratio of the LBTS_ACO between the grid transaction processing system and grid processing system. It is observed that the LBTS_ACO works better when completion time is more than 700 time units. It means the proposed algorithm works better for the transaction processing in the grid environment.

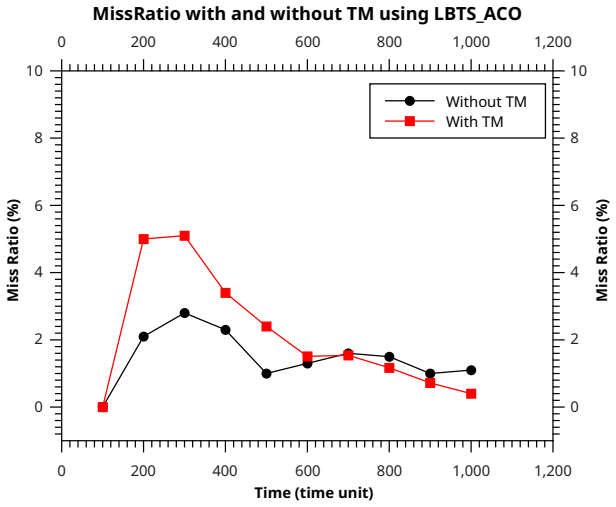

FIGURE 3.17: Comparison of the LBTS_ACO Miss Ratio

3.3.6 Load Balancing Speedup

This experiment compares the load balancing speedup among the mentioned algorithms. The increase in the load balancing speedup increases the overall performance of the grid processing system. **TABLE 3.12** shows the comparison results of the load balancing speedup among the algorithms.

TABLE 3.12: load balancing speedup

Transactions	LBTS_ACO		EO		GA		HLBA		DLB		Randomized	
	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM
100	1.701	2.15	1.7	2.1	1.6	1.8	1.68	1.73	1.19	1.2	0.94	0.94
200	1.602	2.25	1.6	2.2	1.59	2.09	1.53	1.84	1.16	1.17	0.91	0.95
300	1.418	2.09	1.4	2.0	1.399	1.99	1.4	1.64	1.121	1.21	0.88	0.88
400	1.458	1.82	1.457	1.75	1.395	1.7	1.456	1.55	1.07	1.133	0.84	0.89
500	1.425	1.78	1.42	1.75	1.399	1.72	1.348	1.52	1.05	1.06	0.82	0.83
600	1.466	1.83	1.455	1.8	1.4	1.799	1.3569	1.58	1.05	1.03	0.82	0.81
700	1.503	1.58	1.5	1.55	1.45	1.5	1.497	1.47	1.09	1.01	0.86	0.79
800	1.565	1.58	1.5	1.5	1.45	1.49	1.451	1.39	1.07	1.00	0.848	0.78
900	1.597	1.51	1.567	1.5	1.55	1.49	1.48	1.43	1.00	1.02	0.877	0.80
1000	1.6	1.47	1.575	1.47	1.57	1.42	1.46	1.41	1.03	0.826	0.812	0.64

In **FIGURE 3.18** we see the effects of the load balancing speedup under a varied number of the transactions. The LBTS_ACO performs better than other algorithms. The reason is that the proposed algorithm manages the load efficiently as compared to other algorithms.

In **FIGURE 3.19** we have the load balancing speedup comparison of all the algorithms for grid processing system (without the transaction processing). The gaps among the curves

show that the LBTS_ACO works much better than other algorithms. The reason is that the load balancing process of the LBTS_ACO in this scenario works well.

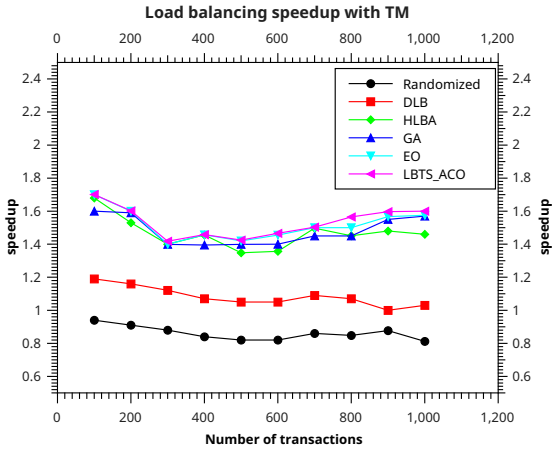

FIGURE 3.18: Load balancing speedup with Transaction Processing

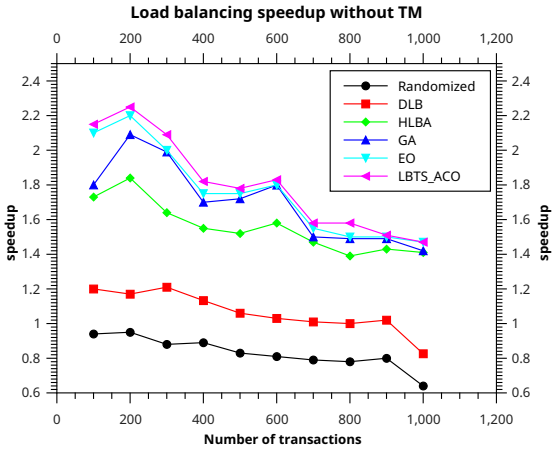

FIGURE 3.19: Load balancing speedup without Transaction Processing

When we compare the load speedup (shown in **FIGURE 3.20**) of the LBTS_ACO between the grid transaction processing system and grid processing system, the load balancing speedup in the grid processing system for first 800 number of the transactions is much higher than in the grid transaction processing system. After 800 number of the transactions, the load balancing speedup of the grid transaction processing system becomes higher than grid processing system.

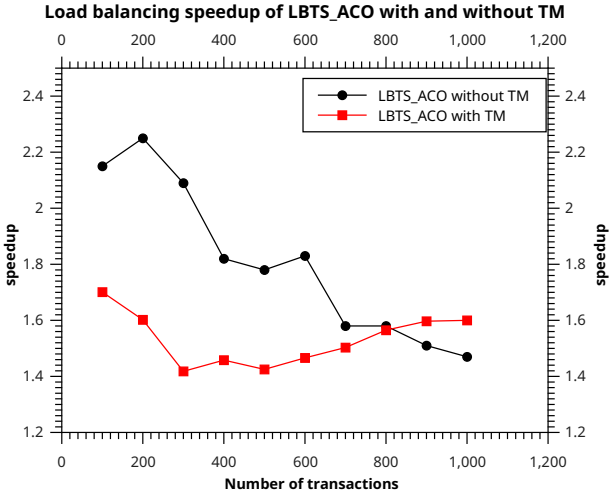

FIGURE 3.20: Comparison of the LBTS_ACO load balancing speedup

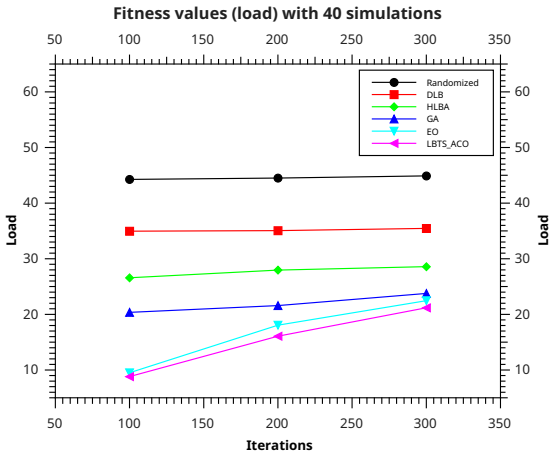

FIGURE 3.21: Load on the system for 40 simulations

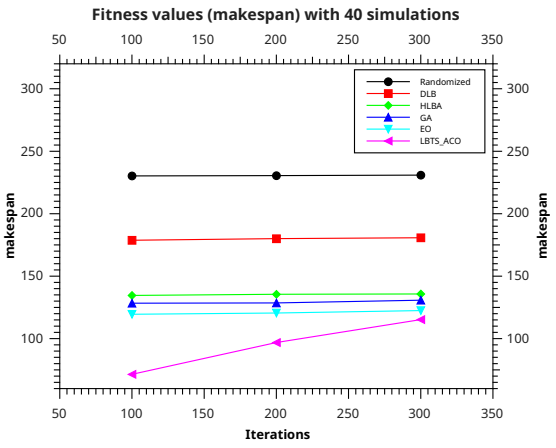

FIGURE 3.22: Makespan results for 40 simulations

3.3.7 Convergence Behaviour

In **FIGURE 3.21** we see the best absolute values of the objective function (load). We see at all the iterations, the LBTS_ACO has less load than others. While in **FIGURE 3.22** we see the best absolute values for the objective function (makespan).

Simulation results presented in this work distinctly show the improvement in the performance of the proposed strategy compared to the other five algorithms.

3.4 Summary

The balanced transaction scheduling in grid is an NP-hard problem. To solve such problems, the meta-heuristic approaches are considerably best options. In this chapter, an ACO based balanced the grid transaction scheduling algorithm, LBTS_ACO, is presented. The algorithm balances the load of the system before the transactions are scheduled to the required nodes. It maximizes the node utilization, minimizes the makespan, increases the throughput, minimizes the miss ratio, and maximizes the load balancing speedup of the grid transaction processing system. The traditional scheduling approach in grid transaction processing system can not do well due to the variations in load and nature of heterogeneity in the grid. To deal with the situations, LBTS_ACO is

the alternative approach which schedules transactions based on the load on the nodes. It is a dynamic type of scheduling algorithm. The algorithm gives better throughput as well as makespan for long-lived transactions as compared to other existing scheduling algorithms used in grid computing systems. We compared the proposed algorithm with the existing algorithms such as EO, GA, DLB, HLBA, Randomized. We also compared the performance of the algorithms on transaction processing system with those on non transaction processing system.

For future work, the next step would be to model and analyze the availability, reliability, and dependability of the data grid system by meta-heuristic transaction scheduling algorithms.