Chapter 5

Load Balanced Scheduling considering Performability

This chapter presents a Honey Bee Optimization (HBO) based method [106] to solve the problem of scheduling of load balanced transactions. In this method, first, the load of the system is balanced and then the transactions are scheduled using foraging behavior of honey bees to find the optimal solutions. We also modify four known scheduling algorithms such as Ant Colony Optimization (ACO), Hierarchical Load Balanced Algorithm (HLBA), Dynamic and Decentralized Load Balancing (DLB), and Randomized to obtain transaction scheduling algorithms for the purpose of comparison with our proposed algorithm. The compared results show that the proposed algorithm performs better than the modified existing algorithms.

5.1 Load Balanced Transaction Scheduling Model

The load balanced transaction scheduling model in on-demand computing environment is depicted in **FIGURE 5.1**. The model consists of six modules. The model first generates a set of transaction requests T with their deadline d_i $(i = 1,2,\ldots,m)$. Each transaction T_i executes within its assigned deadline, otherwise, it will be rolled back or is aborted. The load balanced transaction scheduling is applied to find the appropriate node on a set of N nodes N_j $(j = 1,2,\ldots,N)$. When transactions is executed within their prescribed

deadline, the number of successful transactions are increased. These successful transactions are sent 'committed' and unsuccessful ones are sent 'rolled back'. If a transaction faces rollback and deadline have not expired, the transaction is again sent back to the new optimal processing node. Lastly, the availability and performance of the system are measured. This is measured by Performability Analyzer in **FIGURE 5.1**. How each module of this model works is given below:

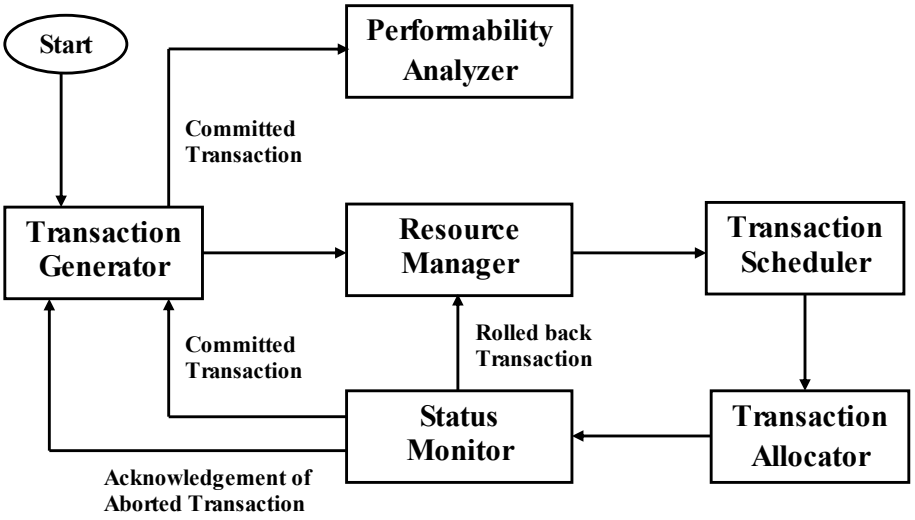


FIGURE 5.1: Performability-Aware Load Balanced Transaction Scheduling Model

- Transaction Generator: In this module, the transaction requests are generated. Transactions arrive in Poisson fashion to the system with the rate λ. The resource manager, then, finds out suitable resources for these transactions. After completion of the whole process of the transaction, it receives either the status of committed transactions or the acknowledgment of aborted transactions.

- Resource Manager: This module manages the pool of resources available to the Grid, i.e., the scheduling of the nodes, network bandwidth, and disk storage. It manages the resources for rolled back transactions also.

- Transaction Scheduler: The transaction scheduling which is the important part of the system is accomplished in this module. It maps of transactions to their suitable resources available in the system. But, when the system workload grows heavily, the number of transactions waiting in the queue increases. In this situation, the

scheduler needs to perform load balancing also, over these transactions from a holistic perspective.

- Transaction Allocator: This module selects the load balanced optimal nodes based on the scheduling information. These transactions can access those nodes after the lock is granted. Then transactions get allocated to the scheduled nodes.
- Transaction Monitor: As the name signifies, the module checks or monitors whether transactions have executed within their deadline. If they have, they are allowed to commit or rollback. Otherwise, they are aborted. Thus, the transactions commit, rollback or abort, is decided by Transaction Monitoring phase by using their respective deadlines.
- Performability Analyzer: The module analyses the performability of on-demand computing based transaction processing system. Performability here is the combined analysis of availability and performance which first analyze the availability of the resources and then computes the performance.

The objective of the model is to find an optimal schedule of transactions from the set T to the balanced nodes of the set N with optimized the performance criteria.

5.2 Problem Formulation

In load balanced transaction scheduling problem, m number of transactions and N number of nodes are considered here. Each transaction needs to be completed within their given deadline on one of the nodes. So, the aim of this problem is to find the maximum number of successful transactions that meet their given deadline. Consider the set T of m transactions $T_i(i = 1,2,\ldots,m)$ to be processed within their given deadline, each of them on one node, on the set of N nodes $N_j(j = 1,2,\ldots,N)$. All the transactions can be processed on any of the N nodes. We assume that the completion times of each transaction, t_{T_i} , are independent of the node processing it. The formulations of some important parameters of the problem are given as follow:

Load is the mean number of transactions waiting to be processed in on-demand computing environment. It is estimated as the sum of the mean number of transactions waiting for

each of on-demand computing resources in the infinite execution of the system. Each of on-demand computing resources can be modeled as an M/M/1 queuing system. The steady-state analysis of the M/M/1 queuing systems may result in the mean length of the underlying waiting queues [113]. The average waiting time [114] at each node is given by

$$W_j = \frac{1}{\mu_j - \lambda_j} - \frac{1}{\mu_j} \tag{1}$$

Thus by Little's theorem [114], the mean number of transactions waiting in the queue of the node N_j can be obtained as

$$L_{N_j} = \lambda_j W_j = \frac{{\lambda_j}^2}{\mu_j(\mu_j - \lambda_j)}, \forall j; 1 \leq j \leq N, 0 \leq \lambda_j \leq \mu_j \tag{2}$$

where λ_j is the arrival rate of a transaction and μ_j is the processing rate of transaction at j^{th} node respectively.

Therefore, the sum of the mean number of tasks waiting to be processed in on-demand computing environment can be obtained as

$$L = \sum_{j=1}^{N} \frac{{\lambda_j}^2}{\mu_j(\mu_j - \lambda_j)} \tag{3}$$

Then the processing time of each node can be calculated as

$$PT_j = \frac{L_{N_j}}{\mu_j} \tag{4}$$

Therefore, the processing time of the system can be calculated as

$$PT = \sum_{j=1}^{N} \frac{L}{\mu_j} \tag{5}$$

Thus, the standard deviation of load [82] can be calculated as

$$\sigma = \sqrt{\frac{1}{N} \sum_{j=1}^{N} (PT_j - PT)^2} \tag{6}$$

Availability in on-demand computing based transaction processing system is defined in [18] as the probability that at any time a required minimum fraction of transactions are finished within a given deadline. Thus, the system is said to be available at time t, if the expected fraction of arriving transactions in the small interval $(t, \Delta t)$ which miss their deadlines is less than a given quality of service requirement ϕ. The following availability formulation uses the concept of [18].

If $D(t)$, a random variable, denotes the response time by a transaction at time t, the system is said to be available at time t if

$$Pr[D(t) \leq d] \leq \phi \tag{7}$$

Then probability, $w_d(t)$, that the system is available at time t is given by

$$w_d(t) = Pr[D(t) \leq d] \tag{8}$$

Then availability at time t can be expressed as

$$A(t) = Pr[w_d(t) \leq \phi] \tag{9}$$

If the system is in state S_i, the probability that the response time of a transaction is less than d at the time of arrival of the transaction is given by $w_{d|i}(t)$. Let $r_i(t)$ is the function which denotes the correctness characteristics of the system in the state S_i at time t. Then,

$$r_i(t) = \begin{cases} 1, & \text{if } w_{d|i}(t) \leq \phi \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

Since the transactions arrive in Poisson fashion with rate λ and μ be the transaction processing rate. It is assumed that the failure, repair and processing times of transactions are exponentially distributed. The definition accounts for failure caused not only due to failures of the servers, but also the temporary degradation of performance due to transient overloading of the system. Let γ be the failure rate, and η be the repair rate of the servers. If system is in state S_i, where $S_i = i$ is the number of transactions in the system, and N be the number of servers available at time t and $k \in N$, then $w_{d|k}(t)$ is

given as

$$w_{d|k}(t) = \begin{cases} w_{exp}(d), & \forall k < N \\ (w_{erl} \bigotimes w_{exp})(d), & \forall k \geq N \end{cases} \tag{11}$$

where $w_{exp}(d) = 1 - e^{-\mu d}$ if $k < N$ and $w_{erl}(d) = 1 - \sum_{k=0}^{N-k} \frac{(N\mu d)^k}{k!} e^{-N\mu d}$ if $k \geq N$. Here $\bigotimes$ represents the convolution operator.

If the system be modeled as $M/M/N$ [114], the steady-state probabilities for the model are given by

$$\Pi_0 = \left[\sum_{k=0}^{N-1} \frac{(N\rho)^k}{k!} + \frac{(N\rho)^N}{N!(1-\rho)} \right]^{-1} \tag{12}$$

$$\Pi_k = \frac{(N\rho)^k}{k!} \Pi_0, \quad 1 \leq k \leq N-1 \tag{13}$$

$$\Pi_k = \frac{N^N \rho^k}{N!} \Pi_0, \quad k \geq N \tag{14}$$

where $\rho = \frac{\lambda}{N\mu}$.

Suppose M is the threshold after which all r_i's with $i \geq M$ will be 0. Then, the conditional steady-state availability[1] is given by

$$A_{k,\lambda} = \sum_{i=0}^{M} r_i \Pi_i \tag{15}$$

Now, suppose Q_k be the probability that there are exactly k out of N servers available. If $q = \frac{\eta}{\gamma+\eta}$ be the availability of a single server, then

$$Q_k = \frac{N!}{k!(N-k)!} q^k (1-q)^{N-k} \tag{16}$$

Then availability of the system under certain load λ is given by

$$A_\lambda = \sum_{k=1}^{N} A_{k,\lambda} Q_k \tag{17}$$

[1] Steady-state availability of a system at time t is the probability that the system is functioning correctly when time tends to infinity. Then steady state availability is computed as $A(\infty) = \lim_{Time \to \infty} \frac{1}{Time} \int_0^{Time} A(t)dt$.

where $\forall k = 1,...,N$. Therefore, under a transaction scheduling $X = \{x_{ik}\}_{1 \leqslant i \leqslant m, 1 \leqslant k \leqslant n}$,

$$A_\lambda(X) = \sum_{k=1}^{N} \sum_{i=0}^{M} r_i \Pi_i . \frac{N!}{k!(N-k)!} q^k (1-q)^{N-k} \tag{18}$$

The probability that a transaction successfully completes by the specified deadline is defined as the performability of the transaction. If $P(T)$ be the performability [115] of on-demand computing based transaction processing system, then

$$P(T) = \prod_{i=1}^{m} P_{d(T_i)} \tag{19}$$

where $P_{d(T_i)}$ is the probability that the transaction T_i completes within $d(T_i)$ that may also permit retries if necessary using slack time. Now, if $P_{d(T_i)}$ is the probability that the transaction T_i completes within the estimated execution time $d(T_i)$, then probability $P_{d(T_i)_{retry}}$ that the transaction T_i will meet the deadline, based on retries is given by

$$P_{d(T_i)_{retry}} = \sum_{j=0}^{k-1} (1 - P_{d(T_i)})^j * P_{d(T_i)} \tag{20}$$

where $k * d(T_i) \leq d(T_i)_{retry}$, and $(1 - P_{d(T_i)})$ be the probability that transaction T_i fails to complete within execution time $d(T_i)$. Then the performability in **Eq. (19)** will change to

$$P(T) = \prod_{i=0}^{m} P_{d(T_i)_{retry}} \tag{21}$$

Miss Ratio [7] is used to measure the performance of the algorithm used for transaction execution. It is calculated as

$$Miss\ Ratio = \frac{T_{miss}}{T_{total}} * 100\% \tag{22}$$

where T_{miss} is the number of transactions missing the deadlines and T_{total} is the total number of handled transactions.

We propose the load balanced scheduling in this chapter so that the waiting time on each node can be minimized and it can

1. maximize the resource availability,
2. maximize the performability of the transaction $(P(T))$, and
3. minimize the miss ratio.

5.3 Proposed Algorithm

The section proposes LBTS_HBO (see **Algorithm 3**). This optimization algorithm is based on a foraging behavior of honey bees to find the optimal solution of the problem. Here a transaction represents a honey bee. The set of transactions is the honey bee colony.

- **Load balanced scheduling decision**: The algorithm makes decisions for sending the arriving transactions by load balancing technique. Suppose the system has some threshold limit for load. For this, two possible situations may arise. In the first situation, if the load is less than the threshold value, then the system is balanced. Otherwise, the system is overloaded in the second situation.
- **Transaction scheduling**: After checking the load of the system, the scheduling algorithm selects the appropriate node based on the load value of the node. The node having the lesser value of load is selected.
- **Update of fitness value**: After selecting the appropriate node, the algorithm updates the load value of the selected node and subsequently the overall load of the system is also changed.

The section proposes the LBTS_HBO algorithm. How the LBTS_HBO works is described below.

Suppose T_i of a set of transactions T is the transaction which is arriving in scheduling queue. The scheduler then finds the suitable node N_j from a set of nodes N to assign the transaction T_i. Therefore T_i is the input to this algorithm.

First of all, line 1 initializes node population N. Until the scheduling queue is not empty, lines $2-23$ run **while** loop repeatedly selecting the random nodes to search the optimal

Algorithm 3 LBTS_HBO

INPUT: Transaction T_i
1: Initialization node population ▷ Initialization
2: **while** Scheduling queue is not empty **do**
3: **for** each Transaction T_i in T **do**
4: **for** each node N_j in N **do**
5: **if** $(\sigma \leq \sigma_{thres})$ **then**
6: **if** $(L_{N_j} \leq \lfloor \frac{N}{2} \rfloor)$ **then**
7: Assign T_i to N_j ▷ Assignment of Transaction T_i to node N_j
8: Increment the load of N_j by one ▷ Incrementation of load on node N_j after assignment of transaction
9: **else**
10: Select two random nodes N_{r1} and N_{r2}
11: Compare their loads
12: Select least loaded node out of these two
13: Assign T_i to least loaded node
14: Increment the load of least loaded node by one
15: **end if**
16: **else**
17: Abort transaction T_i
18: **end if**
19: **end for**
20: Update the fitness value to all nodes ▷ Updation of fitness value
21: Find σ of the system
22: **end for**
23: **end while**
OUTPUT: Scheduling status

node for the requested transaction. In each iteration of the while loop, the algorithm performs the following operations. Lines $3-22$ run the **for** loop for each T_i. Lines $4-19$ again run a **for** loop, but this time for each node N_j from the population N. Line 5 checks σ. Two conditions arise in this respect.

1. $\sigma \leq \sigma_{thres}$: If this situation arises, then the system is balanced. Here σ_{thres} lies in $[0,1]$.

2. $\sigma > \sigma_{thres}$: If this situation arises, it means the system is already overloaded and no node is ready to accept more load. Therefore, transactions are aborted instead of assigning to the overloaded nodes.

If the condition is true, then the system is balanced and the load balanced scheduling is triggered. For load balanced scheduling lines 6 to 15 work as follows: Line 6 checks whether $L_{N_j} \leq \lfloor \frac{N}{2} \rfloor$. If it is, then T_i is assigned to N_j in line 7 and load on N_j is incremented by one in line 8. Otherwise, two random nodes are selected randomly from the set of nodes N (line 10). Their loads are compared (line 11). In line 12, least loaded node between two is selected for the assignment of T_i. Line 14 then increments the load of the selected node by one. Whereas, the false condition of line 5 starts in line 17 which says that if $\sigma > \sigma_{thres}$ then system is said to be overloaded and load balanced scheduling is not

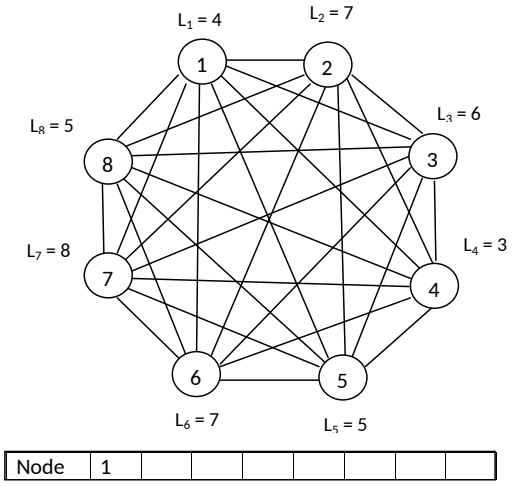
possible. Therefore, it is better to abort the transaction. Then line 20 updates the fitness value to all nodes. Here fitness value means the status of load and capacity of each node. Line 21 finds out the value of σ of the system. We repeat the iterations of the **while** until all the transactions are not scheduled.

5.4 Applying LBTS_HBO Algorithm

The characteristics of balanced scheduling in on-demand computing based transaction processing systems are different from other scheduling in the aspect that the solution is an unordered subset of balanced nodes. In our study model, on-demand computing based transactions are represented by a complete undirected graph. Let $G = (N, E)$ denote the complete undirected graph representing transactions, where N is the set of nodes; $E = N \times N$ is the set of edges between the nodes. **FIGURE 5.2** gives an illustrative example how LBTS_HBO algorithm works. Suppose there are m number of transactions (T_1, T_2, $\ldots$, T_m) which arrive at the system with available nodes (suppose $N = 8$).

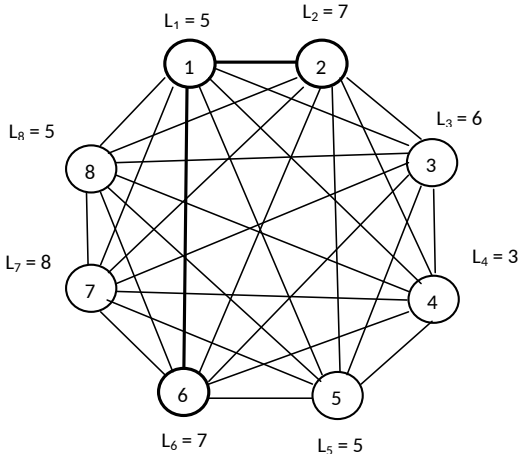
Load balanced scheduling decision: Before going to scheduling decision, the LBTS_HBO constructs a set of the feasible solution so that the scheduler gets the optimal nodes for scheduling incoming transactions. Construction of the solution starts from the initial node N_1. The scheduling decisions are taken as follows:

Suppose, each node has the processing capacity i.e., $\mu_j = 8$. Then total processing capacity of the system is 64. If $L_{N_j} = \{4,7,6,3,5,7,8,5\}$ as shown in **FIGURE 5.1(*a*)**. Then σ will be calculated using **Eq. (6)** as 0.456892. Here σ is calculated using **Eqs. (1)** to **(6)**. If we assume $\sigma_{thres} = 1$. Here $\sigma < \sigma_{thres}$. As subfigure (a) in **FIGURE 5.2** illustrates, the LBTS_HBO 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, N_1 is selected as the best-so-far solution. The load value is incremented by 1, and it becomes 5. Then the bee in the algorithm comes out of the **while** loop. The last position of the bee is still at the node N_1. Therefore, again line 5 of the algorithm checks the condition. This time, the condition fails, so two random nodes N_2 and N_6 are selected as shown in subfigure (b) in **FIGURE 5.2**. Their loads are 7 and 7 respectively. Load on both of the nodes are same, then any of them can be selected. Suppose N_2 is selected. The load value of N_2 is incremented by 1



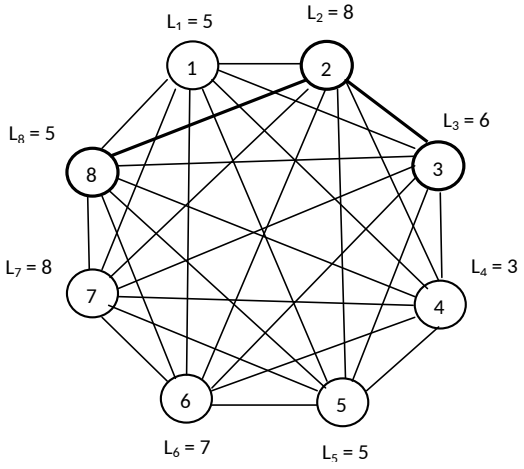
Node	1							

(a)



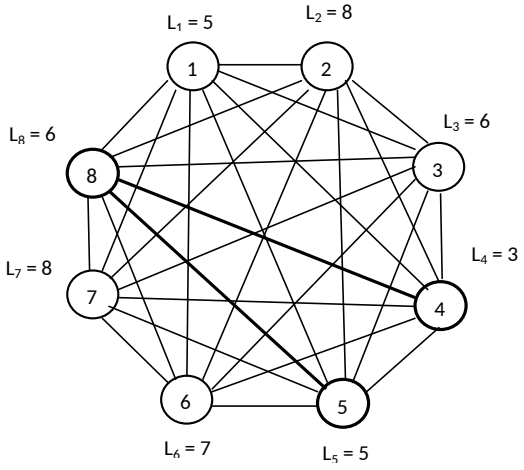
Node	1	2						

(b)



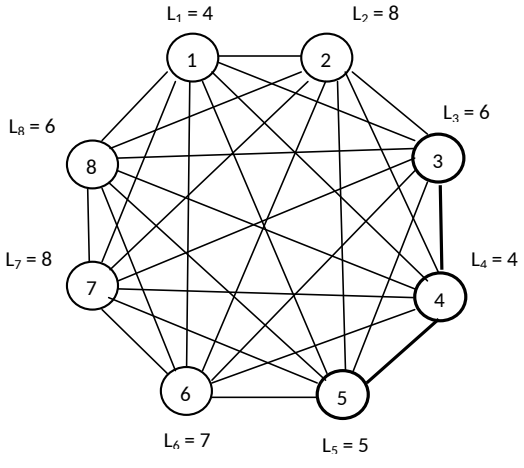
Node	1	2	8					

(c)



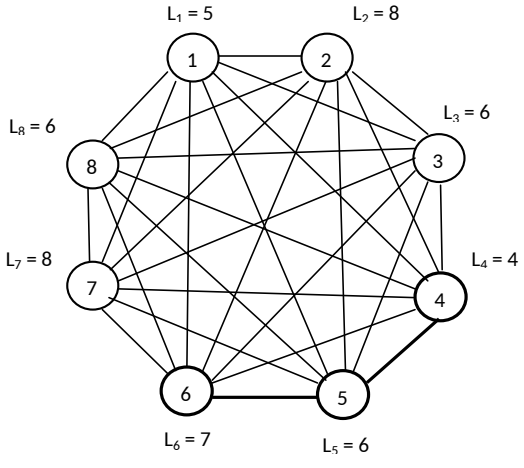
Node	1	2	8	4				

(d)



Node	1	2	8	4	5			

(e)



Node	1	2	8	4	5	4		

(f)

FIGURE 5.2: Working example of the LBTS_HBO when N=8

and it becomes 8. In subfigure (c), the bee is at N_2. Line 6 checks the condition. Since it is false, two random nodes N_3 and N_8 with their respective load 6 and 5 are selected. The N_8 is least loaded between N_3 and N_8. Here N_8 is selected as the solution. The same procedure continues in subfigure (c) to (f) in **FIGURE 5.2**. The solution is kept in the scheduling queue. At every step from **FIGURE 5.1(*a*)** to **FIGURE 5.1(*f*)**, we found that the $\sigma \leq 1$.

Transaction scheduling: After the scheduling decision is done, the algorithm prepares the scheduling list of transactions. According to this list the transactions are dispatched to their assigned nodes.

Update of fitness value: After each iteration of the **while** loop in the algorithm, the fitness value of all the nodes are changed. Because, at every iteration the load value of selected node is changed which is used in finding out the fitness value of a node. Then the algorithms finds the σ of the system.

5.5 Simulation and Result Analysis

We have chosen Colored Petri Nets (CPNs or CP-nets) for the simulation of our work. 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.

As depicted in **FIGURE 5.1**, created by transaction generator, transactions first enter the transaction queue and then get scheduled in a transaction allocator before arriving at the appropriate node. The peformability of the system is checked every time.

We modify four known scheduling algorithms to obtain transaction scheduling algorithms for the purpose of comparison with our proposed algorithm. We compare the performances of the proposed algorithm with the four scheduling algorithms; ACO [36], Hierarchical Load Balanced Algorithm (HLBA) [116], 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 non-transaction processing system. We thus ran each algorithm 10 times at each

time unit value for every problem instances to get the result. We used term TM for transaction management and WTM for without transaction management in result graphs.

5.5.1 Comparison of Time Complexity of LBTS_HBO Algorithm with Other Algorithms

In each iteration the algorithm LBTS_HBO iterates through all vertices of the graph (see **FIGURE 5.2**). Let us suppose the scheduling queue is of fixed size. The time to be elapsed for completing whole queue is c_1. Line 3 of the algorithm runs **for** loop for each transactions. The number of transactions is assumed as m. Line 4 runs another **for** loop for each node. The number of nodes is assumed as n. Then assignment process starts. Let us suppose, the assignment process takes c_2 time. Then time complexity of the algorithm is $\mathcal{O}(c_1.mn.c_2)$ *i.e.*, $\mathcal{O}(mn)$. Therefore, time complexity of our algorithm LBTS_HBO is $\mathcal{O}(mn)$ where m is the number of transactions and n is the number of nodes in on-demand computing system. We can see the comparison of time complexity with other four algorithms in **TABLE 5.1**.

Along with the time complexity we also have the characteristics of the existing and our proposed algorithm as shown in **TABLE 5.2**.

TABLE 5.1: Comparison of time complexity of our algorithm with other algorithms

Algorithms	Time Complexity
LBTS_HBO	$\mathcal{O}(mn)$
ACO	$\mathcal{O}(mn)$
HLBA	$\mathcal{O}(m.n\log n)$
DLB	$\mathcal{O}(mn^2)$
Randomized	$\mathcal{O}(mn^2)$

TABLE 5.2: Comparison of the characteristics of the existing algorithms with our proposed algorithm LBTS_HBO

Algorithms	Load balancing	Scheduling	Heuristic	Meta-heuristic	Dynamic	Real-time	Transaction Processing
ELISA [13]	√	√	√		√		
HLBA [34]	√	√	√		√		
MFTF [60, 61]	√	√	√		√		
DLB [35]	√	√	√		√		
ECLB [10]	√		√		√	√	
GA [1]	√	√		√	√		
Hybrid real-coded GA [62]	√			√	√	√	
Extremal Optimization [2]		√		√	√		
HBB-LB [82]	√			√	√		
BCO [83]		√		√	√		
MBO [85]		√		√	√		
BLA [86]		√		√	√		
ACO [36]	√	√		√	√		
Randomized Algorithm [36]	√	√	√		√		
LBTS_HBO	√	√	√	√	√	√	√

5.5.2 Availability Analysis of Resources in the System

Resource availability is the important attribute of the system dependability. In this section, we analyze the comparative results of resource availability using the mentioned algorithms as shown in **TABLE 5.3**. The results show that the proposed algorithm works better in on-demand computing based transaction processing system as compared to the computational grid. Minimization of load on each node enhances the resource availability for a deadline-constrained transaction. More and more transactions can be executed successfully when they get completed within their deadlines. But in the case of the computational grid, the jobs may not be time-bound (deadline). Therefore, the proposed algorithm is not so much suitable in the computational grid.

We see in **FIGURE 5.4** which shows the comparative analysis of resource availability when all the algorithms are run in on-demand computing based transaction processing system. The comparison of results shows that both ACO and the proposed algorithm work better than other algorithms. It means the meta-heuristic approaches are suitable in

this condition for the defined problem. However, our algorithm works better than ACO at many instances of time.

We see in **FIGURE 5.5** which illustrates the comparison of resource availability when all algorithms are run in on-demand computing environment without transaction processing. The results show that both the ACO and our proposed algorithms work better than others. However, the proposed algorithm works comparatively better than ACO when the deadline time is less than 400 and greater than 900 time unit.

TABLE 5.3: Resource Availability

Time	LBTS_HBO		ACO		HLBA		DLB		Randomized	
	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM
100	0.9977	0.9977	0.99999	0.9911	0.989	0.9879	0.9879	0.9861	0.9877	0.9857
200	0.9965	0.9977	0.9958	0.9922	0.9890	0.9863	0.9884	0.9857	0.9863	0.9830
300	0.9968	0.9955	0.9932	0.9936	0.9882	0.9872	0.9886	0.9868	0.9881	0.9863
400	0.9939	0.9912	0.9928	0.9933	0.9906	0.9836	0.9910	0.9829	0.9903	0.9822
500	0.9881	0.9903	0.9911	0.9929	0.9887	0.9858	0.9886	0.9857	0.9879	0.9836
600	0.9896	0.9882	0.9911	0.9915	0.9885	0.9872	0.9888	0.9867	0.9881	0.9845
700	0.9931	0.9884	0.9905	0.9887	0.9839	0.9843	0.9841	0.9839	0.9836	0.9836
800	0.9887	0.9873	0.99	0.9886	0.9855	0.9849	0.9859	0.9846	0.9856	0.9842
900	0.9890	0.9890	0.9894	0.9884	0.9842	0.9863	0.9840	0.9860	0.9839	0.9857
1000	0.9895	0.9895	0.9879	0.9890	0.9879	0.9868	0.9876	0.9863	0.9871	0.9854

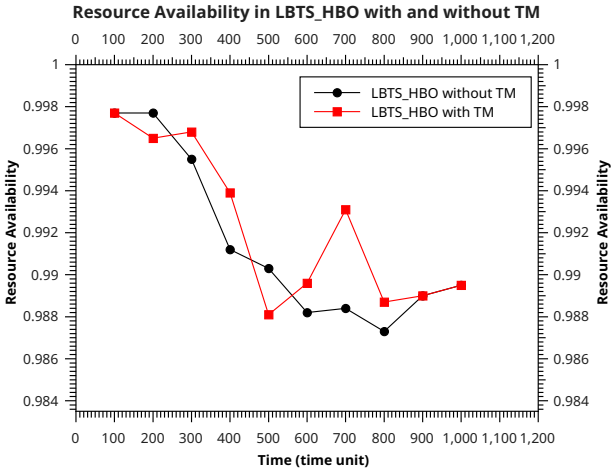


FIGURE 5.3: Resource Availability in LBTS_HBO with and without transaction management (TM)

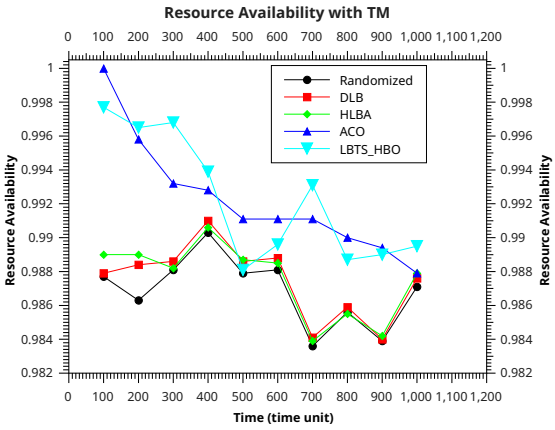


FIGURE 5.4: Resource Availability when transaction management (TM) is used

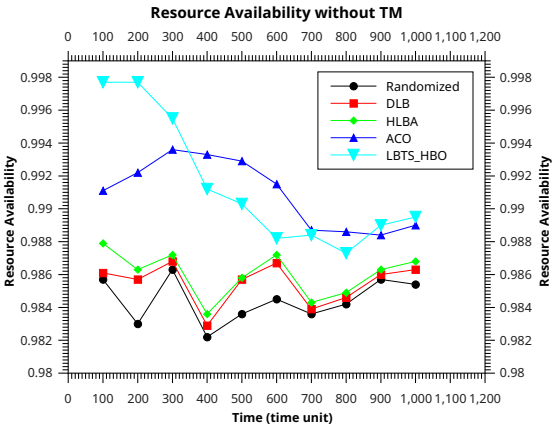


FIGURE 5.5: Resource Availability when no transaction management (TM) is used

5.5.3 Performability Analysis of the System

This section presents the performability analysis of on-demand based transaction processing system. Performability is a composite measure of a system's performance and its dependability. This measure becomes the vital evaluation method for on-demand computing systems which are highly dependable systems, because the systems undergo a graceful degradation of performance in the presence of faults (malfunctions) allowing continued "normal" operation.

We have the comparative results of performability using the mentioned algorithms as shown in **TABLE 5.4**. The results show that the LBTS_HBO performs better than the others. We see in **FIGURE 5.6** which shows how our proposed algorithm works in on-demand computing based transaction processing system as well as in on-demand computing system without transaction processing. It is evident that the algorithm works better in on-demand computing based transaction processing compared to the computational grid. The reason behind this is that the algorithm is designed for deadline-constrained transactions. The deadline condition, if applied in the computational system which requires a long time to execute the larger jobs, the output of successful jobs will see the decline with the limited time-bound as a deadline. For the computational grid, the algorithm works well only for those jobs which require shorter execution time (less than 400 time unit). The results prove that the algorithm is suitable

for on-demand computing based transaction processing system. Instead of this we also compare other existing algorithms.

The comparative analysis of performability is shown in **FIGURE 5.7** when all the algorithms are run in on-demand computing based transaction processing system environment. We compare our algorithm with existing four algorithms. It is evident from the results that the proposed algorithm outperforms the other algorithms. We have the comparative analysis of performability when transaction processing is missing in on-demand computing system as shown in **FIGURE 5.8**. Here we see that the HLBA performs approximately same as our algorithm. Because our algorithm does not perform better than HLBA when executing the larger computational jobs as compared to transactions having their respective deadlines.

TABLE 5.4: Performability

Time	LBTS_HBO		ACO		HLBA		DLB		Randomized	
	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM
100	0.667	0.667	0.999	0.2	0.6436	0.6153	0	0	0	0
200	0.865	0.926	0.77	0.562	0.923	0.913	0	0	0	0
300	0.953	0.93	0.775	0.823	0.9426	0.9456	0.864	0.816	0.833	0.778
400	0.949	0.893	0.895	0.874	0.9651	0.9551	0.9105	0.8641	0.889	0.833
500	0.977	0.94	0.915	0.900	0.975	0.9691	0.9228	0.9097	0.92	0.886
600	0.95	0.942	0.927	0.932	0.9796	0.9782	0.955	0.9507	0.9432	0.938
700	0.97	0.943	0.935	0.93	0.9781	0.9818	0.9605	0.9648	0.9503	0.9557
800	0.96	0.954	0.94	0.937	0.9854	0.9831	0.955	0.9697	0.9482	0.962
900	0.969	0.964	0.95	0.94	0.9838	0.9872	0.9704	0.9746	0.9625	0.9678
1000	0.973	0.964	0.955	0.954	0.9892	0.9875	0.9765	0.9784	0.9702	0.9727

5.5.4 Miss Ratio of Transactions in the System

This section illustrates the comparative analysis of miss ratio of transactions when the mentioned algorithms are applied in on-demand computing based transaction processing as well as in the computational grid.

We have **TABLE 5.5** which depicts the comparative results of the simulation. The results show that miss ratio is minimized when the proposed algorithm is applied. The minimization of miss ratio indicates that the number of successful transactions are increased which is one of our objectives.

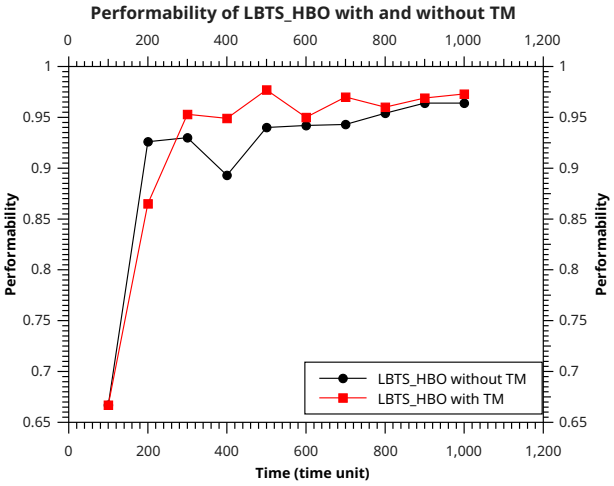


FIGURE 5.6: Performability in LBTS_HBO with transaction management (TM) and without transaction management

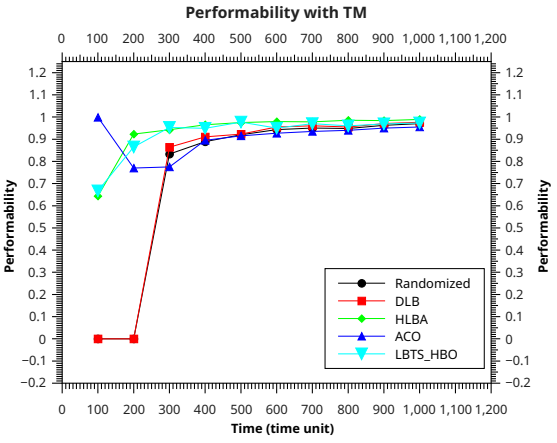


FIGURE 5.7: Performability when transaction management is used

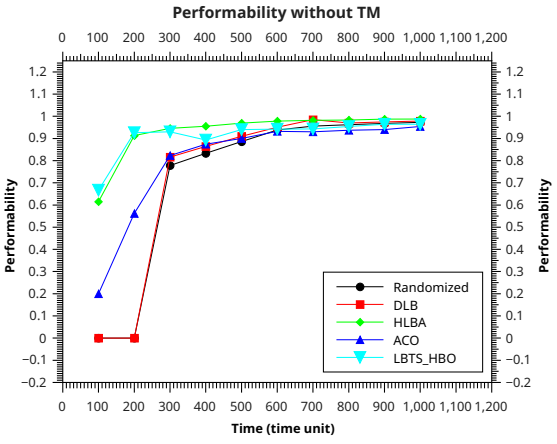


FIGURE 5.8: Performability when no transaction management is used

We see in **FIGURE 5.9** which illustrates the miss ratio comparison when the proposed algorithms run separately, first in on-demand computing based transaction processing system and later in the computational grid. The result shows that our algorithm works better in transaction processing environment as compared to the computational environment.

We see in **FIGURE 5.10** which depicts the comparative analysis of miss ratio among all the mentioned algorithms. The results show that our algorithm is much better than others at all instances of time. Even in the case of the computational environment (as shown in

FIGURE 5.11), the proposed algorithm performs better than others. The improvement in results is caused owing to the balanced scheduling approach of the algorithm. When all the nodes are balanced, the chances of transaction commit is increased, because waiting time at each node is minimized.

TABLE 5.5: Miss Ratio (%)

Time	LBTS_HBO		ACO		HLBA		DLB		Randomized	
	TM	WTM	TM	WTM	TM	WTM	TM	WTM	TM	WTM
100	33.33	33.33	34	35	35.6	33.3	38	40	50	50
200	13.51	7.407	23.07	43.75	30.4	35.6	35	38	39	39
300	4.65	7.05	22.454	17.645	22.8	23.6	53	60	65	68
400	5.140	10.738	10.484	12.631	13.2	17.6	36.2	54	44.4	64
500	6.323	5.99	8.475	9.937	13.6	11.6	25.36	36.8	32	45.6
600	4.987	5.76	7.317	6.785	9.98	11.6	20	20.4	22.4	24.4
700	3.00	5.66	6.472	6.963	8.4	11.5	15.76	14.4	19.6	17.6
800	4.104	4.64	6.00	6.340	8.30	9.50	16.44	12.12	22.4	15.2
900	3.05	3.632	4.914	6.149	6.4	7.4	11.84	11.6	22.4	12.8
1000	2.65	3.728	4.414	4.608	5.4	7.6	9.2	8.6	11.6	10.8

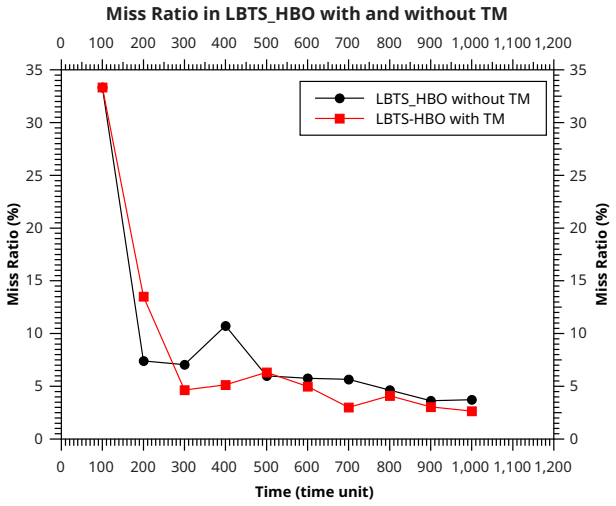


FIGURE 5.9: Miss Ratio in LBTS_HBO with and without transaction management (TM)

5.6 Observations

In this chapter we formulated performability as well as miss ratio of transaction processing in on-demand computing system. Load balanced scheduling of transactions

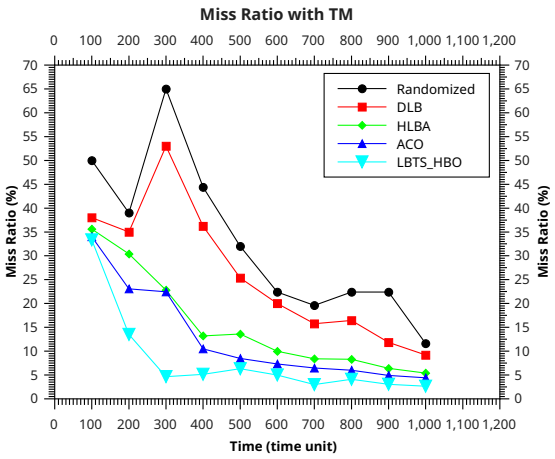


FIGURE 5.10: Miss Ratio when transaction management is used

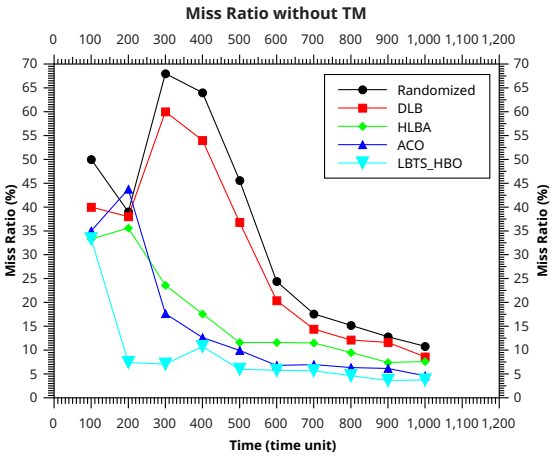


FIGURE 5.11: Miss Ratio when no transaction management is used

in this system to maximize performability is very complex task. In order to solve this problem, we proposed LBTS_HBO algorithm. We compared the performance of our algorithm with ACO, HLBA, DLB, Randomized algorithms. The experimental results show that LBTS_HBO outperforms other algorithms.

5.7 Summary

In this chapter, we proposed a load balanced scheduling technique for on-demand computing based transaction processing systems based on the behavior of honey bee foraging strategy. The algorithm balances the load before scheduling the transactions to appropriate nodes in on-demand computing environment. The transactions are treated as honey bees. This chapter formulates the resource availability and performability considering load. We modified four scheduling algorithms and obtained transaction scheduling algorithms for the purpose of comparison with our proposed algorithm. The result section in this chapter shows that the algorithm enhances the resource availability by decreasing the load. It also increases performability and reduces the miss ratio. This load balanced scheduling algorithm works well for on-demand computing based transaction processing systems. In future, we plan to extend this work to analyze the dependability of the system.