Chapter 4

Transaction Scheduling considering Availability

Maximization of availability for task scheduling in on-demand computing based transaction processing system is an emerging problem. The existing approaches to find the exact solutions for this problem are limited. This chapter proposes a task scheduling algorithm using ant colony optimization (ACO) to solve the mentioned problem. In this method, first, availability of the system is computed, and then the transactions are scheduled using foraging behavior of ants to find the optimal solutions. We also modify two known meta-heuristic algorithms such as Genetic Algorithm (GA) and Extremal Optimization (EO) 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 others.

4.1 Problem Formulation

On-demand computing based transaction processing system consists of a set of various heterogeneous and homogeneous resources which are geographically distributed. To maintain the quality of service (QoS) for the transaction processing in such environment is a challenge. Because, the on-demand computing is a parallel and distributed system and thus there are many issues regarding computing of this system, for example, data locality and availability, scalability, implementation, autonomy, maintenance, fault tolerance, privacy, security are needed to be addressed well before the commercialization of such system. In this chapter, we discuss one of these issues, resource availability.

Transaction scheduling and transaction allocation in on-demand computing system are important strategies which enhances the efficiency of resource management in the system. Transaction scheduling is the method which assigns the transactions to the suitable resources with the purpose to execute them within their prescribed deadline along with optimizing some scheduling parameters. Transaction scheduling in the on-demand computing environment becomes an NP-hard problem as it offers a large search space of possible solutions.

In [1], authors have tried to formulate this problem successfully. But the problem focuses on the resource availability and makespan for task scheduling in grid computing system. Our problem formulation centers on the resource availability and the makespan for transaction scheduling in on-demand computing system. Each transaction must be executed with the consideration of negligible deadline-miss chance. But we start the problem formulation with same sequence as it was given in [1].

Resource availability which is an important issue in maintaining the QoS of any computing system has two key parameters such as failure rate and repair rate. The failure rate is the number of failure per unit time while repair rate is the number of repairs per unit time. Since the on-demand computing system is also a repairable type of system, MTTF and MTTR are used to compute the resource availability of the system.

In this chapter, our addressed problems are the maximization of resource availability and minimization of makespan in the on-demand computing based transaction processing system.

Definition 4.1.1. *The availability of the on-demand computing system nodes (resources) is expressed as the probability that the nodes are available for a given time interval.*

Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR) are the two parameters which are used to compute the availability of the on-demand computing based transaction processing system.

Definition 4.1.2. *MTTF is defined as the expected time between two consecutive failures of a node.*

Definition 4.1.3. *MTTR is defined as the expected time between two consecutive repairs of a node.*

Definition 4.1.4. *Mean time between failure (MTBF) is defined as the average time between two consecutive failures.*

MTBF can be expressed as

$$MTBF = MTTF + MTTR \tag{1}$$

Based on the MTTF, MTTR, and MTBF, the availability, A_t, of j^{th} node for time t, is computed as

$$A_j(t) = \frac{MTTF_j}{MTTF_j + MTTR_j}, \ \forall j = 1, \ldots, n \tag{2}$$

The availability can be expressed in two ways; (1) in series arrangement of nodes, and (2) in parallel arrangement of nodes.

In **series arrangement** of nodes, the availability is computed as

$$A_s(t) = \prod_{j=1}^{n} A_j(t) \tag{3}$$

where the availability depends on the failure of at least one node, $A_s(t)$ denotes the availability of the entire grid system, and n denotes the total number of nodes in the on-demand computing environment.

In **parallel arrangement** of nodes, the availability is computed as

$$A_p(t) = \left(1 - \prod_{j=1}^{n}(1 - A_j(t))\right) \tag{4}$$

where $A_p(t)$ denotes the availability of the grid system when n number of nodes are arranged in parallel.

The basic formulas in **Eq.** (3) and **Eq.** (4) for the availability computation of series and parallel systems can be used in combination to compute the availability of the system having both series and parallel parts (**series-parallel arrangement**). Assuming all nodes are independent, system availability A_{sp} can be computed from the formula:

$$A_{sp} = \prod_{j=1}^{n}[1-(1-A_j(t))] \tag{5}$$

4.1.1 Makespan

In task scheduling problem, makespan is the important parameter for the evaluation of the method.

Definition 4.1.5. *Makespan is defined as the time required in completing the job.*

Makespan can be computed using the queuing theory. Let M/M/c be the queuing model. Assume λ_j be the task arrival rate at the j^{th} node and μ_j be the service rate of the j^{th} node. Using M/M/c, the waiting time $W_j(t)$ at the j^{th} node can be computed as

$$W_j(t) = \frac{\lambda_j}{(\mu_j - \lambda_j)} \tag{6}$$

Let m is the total number of tasks in the on-demand computing system, and L_j is the total number of tasks allocated on the j^{th} machine, x_{aj} is the assignment function of a^{th} task to the j^{th} node or machine and NIT_a is the number of instruction in the a^{th} task. The assignment function x_{ij} is defined as

$$x_{aj} = \begin{cases} 1, & \text{if } a^{th} \text{ task is allocated on the } j^{th} \text{ node} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Then, total time for execution of allocated transactions at the j^{th} node is computed as

$$T_j = \sum_{i=1}^{L_j} \left[\left(\frac{\lambda_j}{(\mu_j - \lambda_j)} + \left(\frac{1}{\mu_j} \right) * x_{aj} \right) * NIT_a \right] \tag{8}$$

where L_j is the load on the j^{th} node.

In order to calculate the total time for execution of allocated transactions at the j^{th} node, all loops in the algorithm must be computable [110]. Because loops are fundamental for the implementation of almost every algorithm. It should be guaranteed that every loop terminates within a specified amount of time. These types of loops are called *bounded loops*. There are two kinds of *bounded loops* [111] such as:

- Loops with specified limit for the number of iterations
- Loops which are bounded by a time limit that must not be overrun at run time.

Limits for both the maximal number of iterations and for time have to be known to make the computation of maximum execution time possible. Here we assign the limits as 1000 for the maximum number of iterations and 1000s for deadline time.

Let $g \in j$, then time taken by the on-demand computing system will be equal to the maximum time taken by any node in the system. Then this time can be computed as

$$T_g = \max_{1 \le j \le n} \left[\sum_{i=1}^{L_j} \left[T_j \right] \right] \tag{9}$$

The best solution can be obtained from many generated solutions as the minimum of all the solutions. Let *popsize* is the total number of solutions generated in the population. The best solution is obtained by our first objective function which is computed as follows:

$$\min^{popsize} \left[\max_{1 \le j \le n} \left[T_g \right] \right] \tag{10}$$

4.1.2 Availability as Fitness Function

If the nodes are in **series-parallel arrangemnt**, the system will not be unavailable unless all the nodes fail. Since, the maximization of availability is our second objective whose function is given as follows:

$$\overset{popsize}{\max}\,[A_{sp}(t)] \tag{11}$$

where $A_p(t)$ is defined in **Eq.** (4).

4.2 The Proposed Model

We propose an algorithm which maximizes the availability of nodes and minimizes the makespan by using scheduling strategy in the on-demand computing environments. The algorithm uses meta-heuristic based task scheduling to solve the problem.

4.2.1 ACO approach

The ACO was inspired by the foraging behavior of real ants. In search of food, initially, the ants explore randomly in the surrounding area of their nest. When a food source is found by an ant, the ant immediately carries some of the food after it evaluates the quality and quantity of the food. The quantity of the deposited pheromone is the guide to other ants what the quality and quantity of the food is. With the help of pheromone trails, the ants can find the shortest paths between their nests and food sources. They apply a probabilistic approach in selecting the path with the highest pheromone trails on the paths. The pheromone trails gradually start to evaporate. The attractive strength gets on reducing. The more time an ant takes to travel down the path and back again, the more time the pheromones have to evaporate. The pheromone evaporation also avoids the convergence to a locally optimal solution. If the pheromone does not evaporate, the paths chosen by first ants would be excessively attractive to the following ones. The idea of the ant colony algorithm to mimic the behavior of ants with simulated ants. Informally, an ACO algorithm can be imagined as the interplay of three procedures [69]:

- **ConstructAntsSolutions:** This procedure manages a colony of those ants, which concurrently and asynchronously visit adjacent states of the problem. They apply a stochastic local decision policy while moving with the use of pheromone trails and heuristic information. In this way, solutions to the optimization problem are built incrementally.
- **UpdatePheromones:** This procedure modifies the pheromone trails. The trails' value either increases, as ants deposit pheromone on the components or the connections they use, or decreases, due to pheromone evaporation.
- **DaemonActions:** This 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 is useful or not.

The objective for using the ACO algorithm is to parallelize search dynamically over constructive computational threads by incorporating information from previously obtained results. For scheduling problem several algorithms based on ACO have been proposed in the literature. They all focus on the pheromone update. Due to the reason of search speed and solution efficiency, premature convergence occurs. We propose an improved ACO algorithm named MATS_ACO in this chapter.

4.2.2 Proposed Algorithm: MATS_ACO

We propose the MATS_ACO algorithm (**Algorithm 2**) for the objective of maximizing the resource availability for task scheduling in the on-demand computing based transaction processing system.

This algorithm is a stochastic search procedure of nodes or resources having less probability of failure and highest probability of repair (if the failure occurs). The central component of the algorithm is the pheromone model [54] which is used to sample the search space probabilistically. The process is a Combinatorial Optimization problem.

Definition 4.2.1. *If τ_j be the pheromone trail deposited on the j^{th} node, iter is the iteration number, and ρ be the given evaporation rate of the pheromone on the node, then τ_j can be updated as*

$$\tau_j(iter+1) \leftarrow (1-\rho).\tau_j(iter) + \frac{\rho}{iter}.\sum_{iter=1}^{K-1} \tau_j, \forall iter \in K.$$

The parameter $\rho \in (0,1]$ is the evaporation rate. It has the function of uniformly decreasing all the pheromone values. From a practical point of view, pheromone 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 4.8**. We assume that the initial value of pheromone is $\frac{d}{log(iter+1)}$, where d is constant. The value of d is given in **TABLE 4.8** and K is the maximum number of iteration.

Definition 4.2.2. *If ρ be the given evaporation rate of the pheromone of an ant active on the node selected, the quality of the selected node is $\eta(N_j)$, where*

$$\eta(N_j) = \rho.\tau_j, \forall N_j \in N.$$

To calculate the quality of the node, first we need to calculate the pheromone. According to **Definition 4.2.1**, we get the initial value of the pheromone as $\frac{d}{log(iter+1)}$. The pheromone τ_j deposited by the ant on the node N_j is calculated as $\frac{0.25}{log(1+1)} = 0.830482024$ where the value of d is taken as 0.25 given in **TABLE 4.8**. The evaporation rate (ρ) of the pheromone is 0.2, then the quality of the node $(\eta(N_j))$ is computed as $\rho.\tau_j$ which is equal to $0.2 \times 0.830482024 = 0.166096405$.

Definition 4.2.3. *If α be the relative importance of the pheromone of the j^{th} node, τ_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*

$$\boldsymbol{p}(N_j|A^p) = \frac{[\tau_j]^\alpha.[\eta(N_j)]^\beta}{\sum_{N_y \in \mathfrak{N}(A^p)}[\tau_y]^\alpha.[\eta(N_y)]^\beta}, \forall N_j \in \mathfrak{N}(A^p).$$

Here $\mathbf{p}(N_j|A^p)$ 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 4.8**. The choice of a solution component $N_j \in \mathfrak{N}(A^p)$ is, at each construction step, done probabilistically on the pheromone model. The probability for the N_j is proportional $[\tau_j]^\alpha.[\eta(N_j)]^\beta$.

In detail, the MATS_ACO algorithm works as follows: When a transaction arrives at a node in the on-demand computing system, an ant is initialized, and it starts working.

Next step is to find the optimal nodes, the set of the feasible solution. At each iteration, exploiting a given transaction, solutions to the problem under consideration are constructed probabilistically. Finally, before the next iteration starts, the transaction update is performed by using some of the solutions.

In detail, the MATS_ACO works as follows: When a transaction arrives at a node in the on-demand computing system, an ant is initialized, and it starts working. Next step is to find the optimal nodes, the set of the feasible solution. At each iteration, exploiting a given transaction, solutions to the problem under consideration are constructed probabilistically. Finally, before the next iteration starts, the transaction update is performed by using some of the solutions.

Suppose the algorithm starts operation at all node N_j. Line 2 calculates availability and makespan of all the node N_j. Time is initialized as $t = 0$ in line 3. Line 4 initialize the iteration number as $k = 1$. Line 5 calculates the pheromone released by the ant at each node as $\tau_j = e^{A_j}$. Line 6 initializes the maximum number of nodes to be traversed by the ant i. Line 7 initialize the parameter $p_0 = 0$ which is used to attain quick convergence of the algorithm.

ConstructAntsSolutions: Construction of optimal solution is the ingredient module of the algorithm. The module assembles the solutions from the finite set of solution component N. The current solution A^k is extended at each construction step by adding a feasible solution component to the set of feasible solutions. 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 $8-47$ repeatedly selecting the random nodes to search the optimal node for the requested transaction.

UpdatePheromones: Line 42 calculates the pheromone value on the node N_j deposited by the ants as depicted in **Definition 4.2.1**.

FIGURE 4.1 shows the working example of the MATS_ACO.

Algorithm 2 MATS_ACO

INPUT: Number of nodes (n), number of ants (m), Range of load (λ), range of processing speed (μ) of nodes, range of task size, MTTF and MTTR for each node in the on-demand computing system, population size, task deadline (d), and number of generation.

Randomly distribute ants on the nodes ▷ Initialize the population
Calculate

- the availability for each node using **Eq.** (2)
- the fitness value of each individual using **Eq.** (11)
- the makespan using **Eq.** (10)

$t = 0$ ▷ time counter
$iter = 1$ ▷ number of iterations
$\tau_j = e^{A_j}$ ▷ τ_j is the initial pheromone trail on each node N_j $\forall j = 1,\ldots,n$
$\mathcal{N}^i = 0$ ▷ $\mathcal{N}^i$ represents the list of nodes traversed by ant i, $\forall i = 1,\ldots,m$
$p_0 = 0$
while ($iter \leq K$) **do**
  for $i \in [1,m]$ **do**
    for $j \in [1,n]$ **do**
      if ($\mathcal{N}^i \leq \mathcal{N}_n^i$) **then** ▷ $\mathcal{N}_n^i$ gives the maximum number of nodes to be visited by ant i
        Generate random number p ($0 \leq p \leq 1$)
        if ($p \geq p_0$) **then**
          Generate random number q ($q \in S_i$)
          $select = q$
          Choose the node $select$ as the next node to move to
          Add $select$ to $\mathcal{N}^i$, Delete it from G_i and S_i
        else
          Compare the probabilities of possible outgoing nodes using **Definition** 4.2.3
          Choose the node having the highest probability p_j^i
          Generate random number $\bar{q}$ ($0 \leq \bar{q} \leq 1$)
          if ($\bar{q} \geq p_j^i$) **then**
            Generate a random number q ($q \in S_i$)
            $select = q$
            Choose the node $select$ as the next node to move to
            Add $select$ to $\mathcal{N}^i$, Delete it from G_i and S_i
          else
            Choose the node with the highest p_j^i value
          end if
        end if
      else
        $j = j + 1$
      end if
    end for
  end for
  Find A_k^+ ▷ A_k^+ is the optimal availability for the iteration k
  if ($A_k^+ > A_{bs}$) **then**
    $A_{bs} = A_k^+$ ▷ A_{bs} is the best availability
  else
    Do not update A_{bs}
  end if
  Update $\tau_j(t)$
  Empty all tabu list (i.e., $\mathcal{N}^i$)
  $t = t + 1$
  $iter = iter + 1$
  $p_k = \frac{log\ k}{log\ K}$
end while
Schedule the tasks
Calculate availability using **Eq.** (11) and makespan using **Eq.** (10).

OUTPUT: Availability and makespan.

4.2.2.1 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 MATS_ACO algorithm as:

$$P_k = \frac{log(k)}{log(K)} \tag{12}$$

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.

4.2.2.2 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 MATS_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 4.8**.

4.2.2.3 Convergence Test

The convergence of the MATS_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 4. *Given **Algorithm 2** that using the pheromone update rule from Definition 4.2.2 for any pheromone value, the following holds*

$$\lim_{t\to\infty} \tau_j(iter) \leq \frac{\tau_j.K}{\rho} \tag{13}$$

where $\tau_j(iter)$ denotes the pheromone value τ_j at iteration $iter$ while K is the maximum iteration.

Proof. At any iteration, the maximum possible increase of pheromone value τ_j is $\eta(N_j)$ if all solution are equal to N_j with a new choice of solution. Therefore, due to evaporation, the pheromone τ_j at iteration $iter$ is bounded by

$$\tau_j \leftarrow (1-\rho)^{iter}.\frac{d}{log(iter+1)} + K.\sum_{iter=1}^{K}(1-\rho)^{K-iter}.\tau_j \tag{14}$$

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

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

Theorem 5. *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 is greater than 0, because it is anyway bounded by maximum pheromone value. Since minimum pheromone > 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).

□

4.2.3 Applying the MATS_ACO Algorithm: Case Study (using NFSNet)

The MATS_ACO conducts to explore the power set of the set of nodes. In our study model, transaction processing in the on-demand computing is represented by NFSNet [112]. It consists of 14 nodes. FIGURE 4.1 gives an illustrative example how the MATS_ACO works. Suppose there are m number of the transactions $(T_1, T_2, \ldots, T_m)$ which arrive at the system with available nodes (Here $N = 14$).

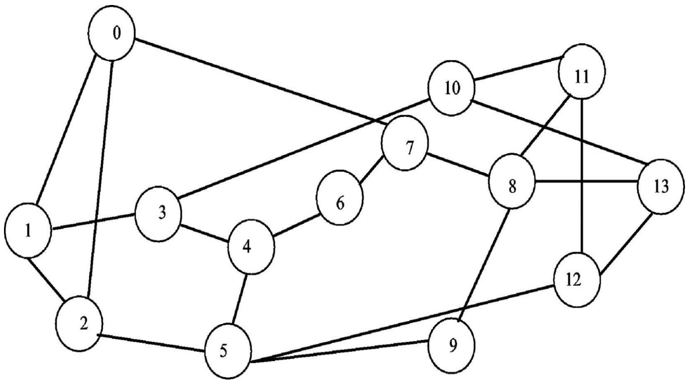

FIGURE 4.1: NFSNet

ConstructAntsSolutions: The MATS_ACO constructs a set of the feasible solution so that the scheduler gets the optimal nodes for scheduling the transactions. Construction of the solution starts from the initial node N_j. As FIGURE 4.1 illustrates, the MATS_ACO works by checking the availability of the node N_j. The value of MTTF, MTTR, and availability of each node is given in TABLE 4.1.

Suppose, $MTTR_j = 10s$ at each node of the graph. At each iteration ants will traverse the nodes and finds out the feasible node which has the highest availability. In this period the

TABLE 4.1: The values of $MTTF_j$, $MTTR_j$, and A_j denote the values of mean time to failure, mean time to repair, and availability of the j^{th} node in NFSNet shown in FIGURE 4.1 in case study II where $N = 14$.

N_j	$MTTF_j$	$MTTR_j$	A_j
N_0	900	10	0.989010989
		50	0.947368421
		100	0.9
N_1	950	10	0.989583333
		50	0.95
		100	0.904761905
N_2	1000	10	0.99009901
		50	0.952380952
		100	0.909090909
N_3	1150	10	0.99137931
		50	0.958333333
		100	0.92
N_4	1100	10	0.990990991
		50	0.956521739
		100	0.916666667
N_5	1200	10	0.991735537
		50	0.96
		100	0.923076923
N_6	1050	10	0.990566038
		50	0.954545455
		100	0.913043478
N_7	1300	10	0.992366412
		50	0.962962963
		100	0.928571429
N_8	1250	10	0.992063492
		50	0.961538462
		100	0.925925926
N_9	1075	10	0.99078341
		50	0.955555556
		100	0.914893617
N_{10}	1175	10	0.991561181
		50	0.959183673
		100	0.921568627
N_{11}	1275	10	0.992217899
		50	0.962264151
		100	0.927272727
N_{12}	975	10	0.989847716
		50	0.951219512
		100	0.906976744
N_{13}	1250	10	0.992063492
		50	0.961538462
		100	0.925925926

ants will release pheromone at each traversed node. In the first iteration, the node N_7 is selected as the optimal solution. In the next iteration, node N_{11} is selected. Similarly, all the nodes are selected based on the availability of the node.

Finally, the scheduling of tasks is conducted using the output of the algorithm. The tasks (transactions) are scheduled based on the list of optimal solutions. The optimal solutions are found as shown in **TABLE 4.2**.

TABLE 4.2: Task scheduling.

Nodes	N_7	N_{11}	N_8	N_{13}	N_5	N_{10}	N_3	N_4	N_9	N_6	N_2	N_{12}	N_1	N_0
Transaction	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}

UpdatePheromones: After selection of the best-so-far solution, the quality of the selected node is calculated, and the ant trails the pheromone on the selected node. In **FIGURE 4.1**, when node N_7 is selected the first time, then pheromone value which is dependent on the initial pheromone and the number of iteration value, i.e., $\frac{d}{log(iter+1)}$ (as depicted in **Definition 4.2.2**) is calculated as $\frac{0.25}{log(1+1)} = 0.830482024$. The quality of the node $\eta(N_7)$ (as depicted in **Definition 4.2.3**) is calculated as $\rho * \tau_7$ where ρ is 0.2 and $\tau_7 = 0.830482024$. Therefore, $\eta(N_7)$ is calculated as 0.166096405. Now the pheromone τ_7 of N_7 is updated as $(1-0.2)*0.830482024 + \frac{0.2}{2}*0.830482024$ which is 0.747433822. This value is deposited at iteration $iter = 2$. As the iteration will keep on increasing; the pheromone will keep on decreasing with evaporation rate $\rho = 0.2$.

DaemonActions: This module of the MATS_ACO updates the set of feasible solution globally by generating a scheduling queue as shown in **TABLE 4.2**.

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.

4.2.4 Time Complexity of MATS_ACO

The time complexity of the proposed algorithm MATS_ACO is calculated as $\mathcal{O}(K.m.n)$ where K is the maximum number of iteration, m is the number of transactions, and n is the total number of nodes in the computing system.

TABLE 4.3: Input parameters for ACO

Parameter	Input
Number of nodes	Minimum 8 Maximum 1000
Number of tasks	Minimum 50 Maximum 1,000,000
Range of load (λ)	$1-100$ MIPS
Range of processing speed (μ)	$101-200$ MIPS
MTTF	$900-1000s$
MTTR	$10-110s$
Population size	50
α	0.5
β	0.5
ρ	0.2
d	0.25

4.3 Experimental Evaluations

In this section, we carried out the a number of experiments by evaluating the proposed algorithms with other two existing alorithms EO and GA. Our work is completely inspired by [1]. The instances parameters used in this chapter is almost similar to [1] which is our base research paper to our work on. Because, we try to implement our proposed algorithm with already given instances and compare our algorithms with the existing algorithms. The proposed scheduling algorithm, MATS_ACO, is evaluated through simulations with Colored Petri Nets (CPNs or CP-nets). 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.

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

- Extremal Optimization: Extremal Optimization (EO) [2] 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 GA [1], 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

TABLE 4.4: Normality Shapiro-Wilk tests for the best results of availability

Data	Shapiro-Wik W	p-value
MATS_ACO	0.889405172	0.0022
EO	0.889830107	0.0024
GA	0.88694925832	0.00258

TABLE 4.5: Wilcoxon statistical tests for the best results (availability) found for ACO, EO and GA algorithms. Assume null hypothesis $\mu_0 = 0$ and null hypothesis: two-sided, $\hat{\mu} < \mu$

Observation	Wilcoxon	p-value	95% Confidence Interval		$\hat{\mu}$
MATS_ACO vs. EO	720	0.351641	$-\infty$	19.1000	-1.09331184504
MATS_ACO vs. GA	730	0.24793775	$-\infty$	17.5000	-1.15169113147

TABLE 4.6: Normality Shapiro-Wilk tests for the best results of makespan

Data	Shapiro-Wik W	p-value
MATS_ACO	0.8668013739216988	< 0.001
EO	0.8642371443799678	< 0.001
GA	0.8558035724664631	< 0.001

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.

We addressed the parameters optimization analysis together with the convergence behavior in section 4.2.2.1, section 4.2.2.2 and section 4.2.2.3. 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 4.4** 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 4.6** shows that the null hypothesis of the samples being in normal distribution must be rejected.

We also conduct nonparametric tests (see **TABLE 4.5**) to check the difference among the methods using Wilcoxon or Mann-Whitney test [109]. The observations shown in **TABLE 4.5** shows that p-value ≤ 0.5. Therefore, it can be concluded that the MATS_ACO outperforms all other algorithms.

TABLE 4.7: Wilcoxon statistical tests for the best results (makespan) found for ACO, EO and GA algorithms. Assume null hypothesis $\mu_0 = 0$ and null hypothesis: two-sided, $\hat{\mu} < \mu$

Observation	Wilcoxon	p-value	95% Confidence Interval		$\hat{\mu}$
MATS_ACO vs. EO	720	0.351641	$-\infty$	19.1000	-1.09331184504
MATS_ACO vs. GA	730	0.24793775	$-\infty$	17.5000	-1.15169113147

4.3.1 Experiment with Varying Mean Time To Failure

The first experiment depicts the effect of MTTF on resource availability in the on-demand computing environment. The input parameters used in the experiment are taken from TABLE 4.8 which is for ACO and TABLE 4.9 and 4.10 show the input parameters for GA used in [1] and EO used in [2] algorithms respectively.

TABLE 4.8: Input parameters for ACO

Parameter	Input
Number of nodes	Minimum 8 Maximum 1000
Number of tasks	Minimum 50 Maximum 1,000,000
Range of load (λ)	$1-100$ MIPS
Range of processing speed (μ)	$101-200$ MIPS
MTTF	$900-1000s$
MTTR	$10-110s$
Population size	50
α	0.5
β	0.5
ρ	0.2
d	0.25

In TABLE 4.8, the values of α, β, and ρ and d has been taken in the experiments. The reason for selecting these fixed values are as follows:

- α is the parameter which is related to the pheromone of the ants released by the ants in the ACO approach. We fixed its value to 0.5, because we tested its values ranging from 0.25 to 10 on the CPU time. We found that when the values of α is increased, the CPU time for the executing the algorithm also increases. But at $\alpha = 0.5$, the algorithm gives optimal result.

- β is the parameter which is related to the quality of the node such as $\eta(N_j)$. We tested that when β is increased, the CPU time for executing the algorithm also increases. The ideal value in this case is $\beta = 0.5$ where we find the optimal solutions.

- The selection of ρ is related to the evaporation rate of the pheromone released by the ants. Here also we tested that when ρ is greater than 0.2, the CPU time increases to find the optimal solution at the maximum iteration 1000. Because, the pheromone trail evaporates too quickly to search the all possible nodes.

- The value of d is related to the initial pheromone released by each ant. It should be initialized in such a manner that it can not affect the speed of selection procedure. If it is higher than 0.25, the CPU time is increased to find the optimal solution. The ideal value for this parameter is fixed as 0.25 in our algorithm.

TABLE 4.9: Input parameters for GA [1]

Parameter	Input
Population size	50
Probability of applying crossover	0.7
Probability of applying mutation	0.05
Probability of applying inversion	0.01

TABLE 4.10: Input parameters for EO [2]

Parameter	Input
Population size	50
Total number of iterations (K)	1000
Probabilistic selection parameter (τ)	0.05
Rank χ	$(0,1)$
γ	$(0,1)$
β	$(0,1)$

When we have the availability observation with different MTTF. It is evident that when MTTF increases, resource availability also increases.

We have the resource availability observation with different MTTF using GA as shown in FIGURE 4.2. When MTTF is between $900-1000s$ then mean availability is 0.768289 and its median is 0.7785, when MTTF is between $1000-1100$ then mean availability is calculated as 0.8126 and its median as 0.8255, when MTTF is between 1100 – 1200 then mean availability changes to 0.833 and its median changes to 0.8455, and when MTTF is between 1200 – 1300, then mean availability is 0.87186 and its median is 0.8779.

TABLE 4.11: The mean and median value of resource availability. FIGURE 4.2 shows the results for GA method while FIGURE 4.3 and FIGURE 4.4 shows the results for EO and MATS_ACO methods respectively.

Strategy	MTTF	mean	median
GA	900 – 1000s	0.768289	0.768289
	1000 – 1100	0.8126	0.8255
	1100 – 1200	0.833	0.8455
	1100 – 1200	0.87186	0.8779
EO	900 – 1000s	0.7858	0.7995
	1000 – 1100	0.8135	0.8185
	1100 – 1200	0.8488	0.845
	1100 – 1200	0.89057	0.8998
MATS_ACO	900 – 1000s	0.84435	0.8693
	1000 – 1100	0.8901	0.92
	1100 – 1200	0.93335	0.9758
	1100 – 1200	0.94607	0.9875

Similarly, we have the resource availability observation with different MTTF using EO shown in FIGURE 4.3. Similarly, FIGURE 4.4 shows the resource availability observation with different MTTF using ACO. All the three methods have different mean and median of resource availability and among them MATS_ACO performs better. The analysis can be seen in TABLE 4.11.

4.3.2 Experiment with Varying Mean Time To Repair

The next experiment depicts the effect of MTTR on resource availability when MTTR varies.

We have the resource availability with different MTTR using GA as shown in FIGURE 4.5. The mean and median of the resource availability are calculated as seen in TABLE 4.12.

We have the resource availability when we use EO algorithm as shown in FIGURE 4.6. FIGURE 4.7 shows the resource availability when we use EO algorithm. From the results,

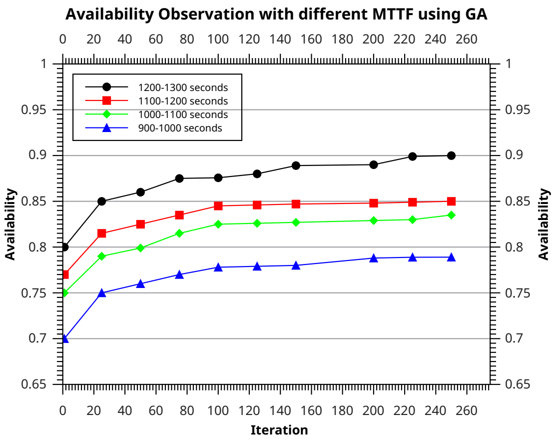

FIGURE 4.2: Availability observation with mean time to failure (MTTF) using GA

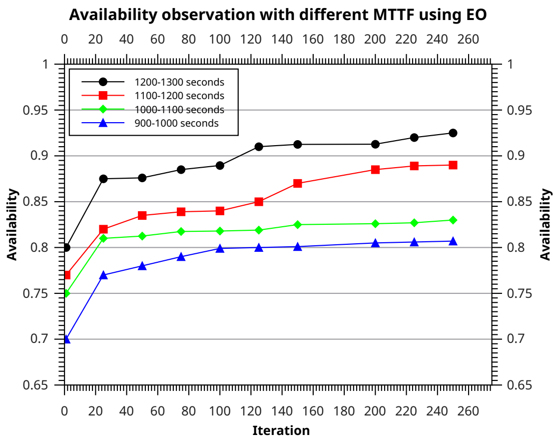

FIGURE 4.3: Availability observation with mean time to failure (MTTF) using EO

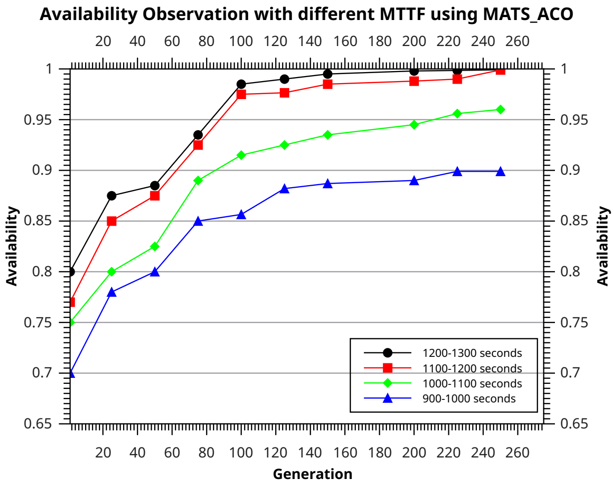

FIGURE 4.4: Availability observation with mean time to failure (MTTF) using MATS_ACO

it is clear that when MTTR increases, resource availability decreases as it takes more time to make the node available.

4.3.3 Experiment with Varying Task Size

In this experiment, we study the effect of task size on resource availability. Tasks are submitted with varying sizes. We consider MTTF for this experiment in the range of $1300 - 1400s$ and task size in million instruction (MI). Other input values are taken from

TABLE 4.12: The mean and median value of resource availability. FIGURE 4.5 shows the results for GA method while FIGURE 4.6 and FIGURE 4.7 shows the results for EO and MATS_ACO methods respectively.

Strategy	MTTR	mean	median
GA	$10-110s$	0.60865	0.7673
	$110-210s$	0.65865	0.7138
	$210-310s$	0.70865	0.6638
	$310-410s$	0.76055	0.6138
EO	$10-110s$	0.66495	0.785
	$110-210s$	0.704	0.7398
	$210-310s$	0.73515	0.7283
	$310-410s$	0.78695	0.6575
MATS_ACO	$10-110s$	0.6823	0.85
	$110-210s$	0.7308	0.8
	$210-310s$	0.78685	0.7388
	$310-410s$	0.83721	0.6875

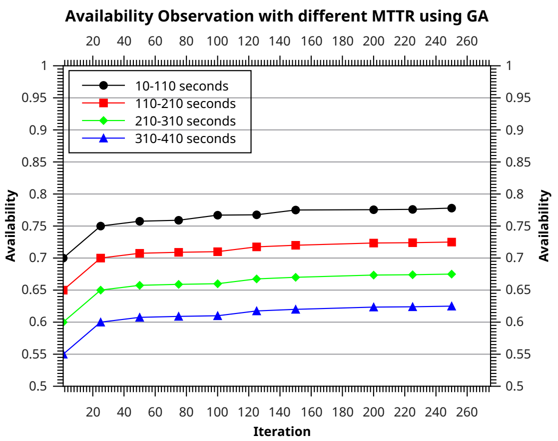

FIGURE 4.5: Availability observation with mean time to repair (MTTR) using GA

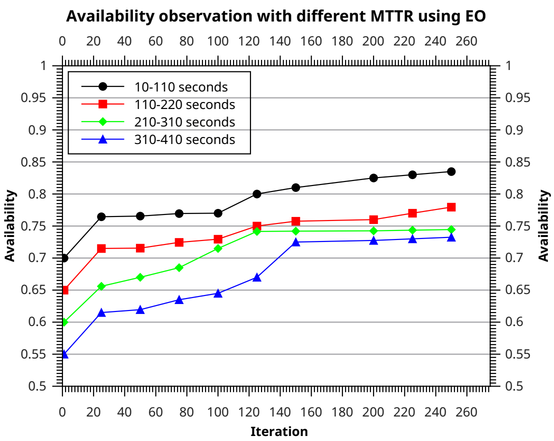

FIGURE 4.6: Availability observation with mean time to repair (MTTR) using EO

TABLE 4.8. We observe that when task size increases, resource availability decreases. We also have figures showing the results with different strategies such as **FIGURE 4.8** with GA method, **FIGURE 4.9** with EO method and **FIGURE 4.10** with MATS_ACO method.

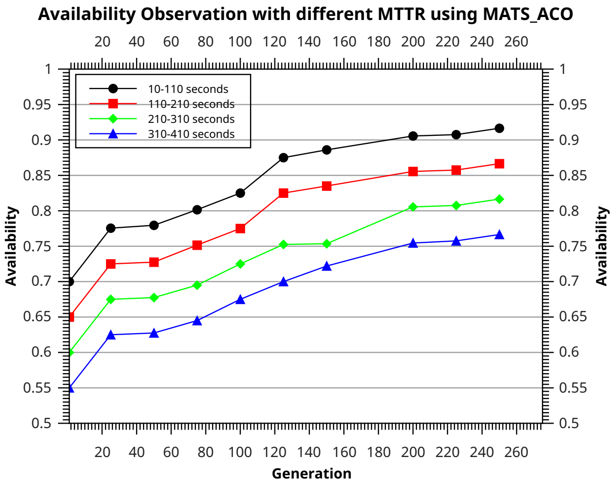

FIGURE 4.7: Availability observation with mean time to repair (MTTR) using MATS_ACO

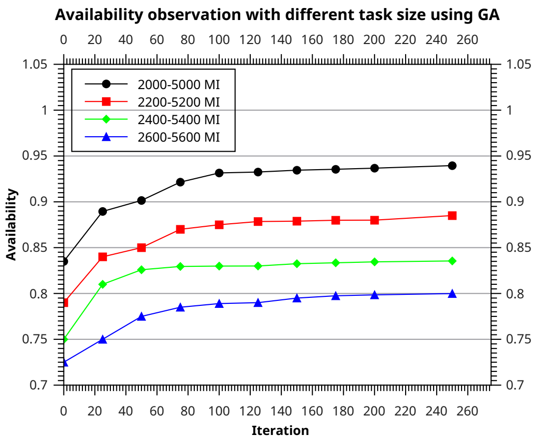

FIGURE 4.8: Availability observation with different task size using GA

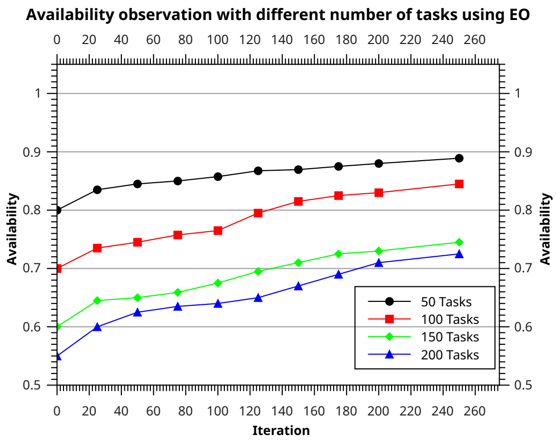

FIGURE 4.9: Availability observation with different task size using EO

4.3.4 Experiment with Varying Number of Tasks

In this experiment, we observe the resource availability by varying the number of tasks in the on-demand computing environment. The input values are taken from TABLE 4.8. It is evident that when the number of tasks increases, the resource availability decreases sharply. Here also we have comparative study of the mentioned algorithms. It is clear that our algorithm works better than others. For result analysis we calculated the mean and median of the output of these algorithms (which is shown in TABLE 4.14).

TABLE 4.13: The mean and median value of resource availability. FIGURE 4.8 shows the results for GA method while FIGURE 4.9 and FIGURE 4.10 shows the results for EO and MATS_ACO methods respectively.

Strategy	Task size	mean	median
GA	2000 – 5000 MI	0.60865	0.6138
	2200 – 5200 MI	0.65865	0.6638
	2400 – 5400 MI	0.70865	0.7138
	2600 – 5600 MI	0.76055	0.7673
EO	2000 – 5000 MI	0.66495	0.6575
	2200 – 5200 MI	0.704	0.7283
	2400 – 5400 MI	0.73515	0.7398
	2600 – 5600 MI	0.78695	0.785
MATS_ACO	2000 – 5000 MI	0.6823	0.6875
	2200 – 5200 MI	0.7308	0.7388
	2400 – 5400 MI	0.78685	0.8
	2600 – 5600 MI	0.83721	0.85

TABLE 4.14: The mean and median value of resource availability. FIGURE 4.11 shows the results for GA method while FIGURE 4.12 and FIGURE 4.13 shows the results for EO and MATS_ACO methods respectively.

Strategy	Number of Tasks	mean	median
GA	50	0.86875	0.877
	100	0.7615	0.7645
	150	0.66195	0.6638
	200	0.6088	0.6148
EO	50	0.85685	0.8625
	100	0.78125	0.78
	150	0.6834	0.685
	200	0.6495	0.645
MATS_ACO	50	0.8773	0.884
	100	0.79375	0.8013
	150	0.7125	0.715
	200	0.664	0.6638

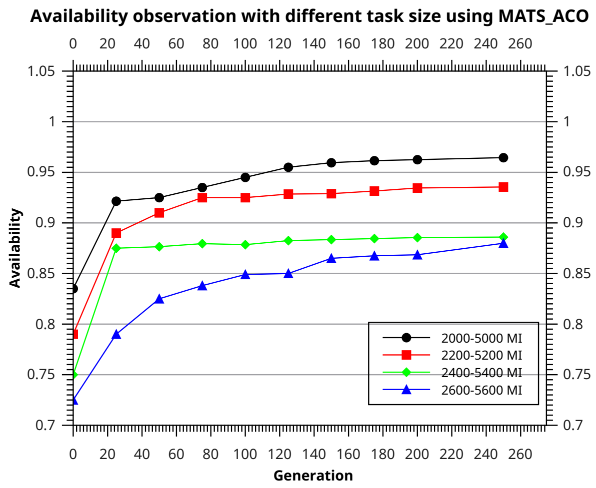

FIGURE 4.10: Availability observation with different task size using MATS_ACO

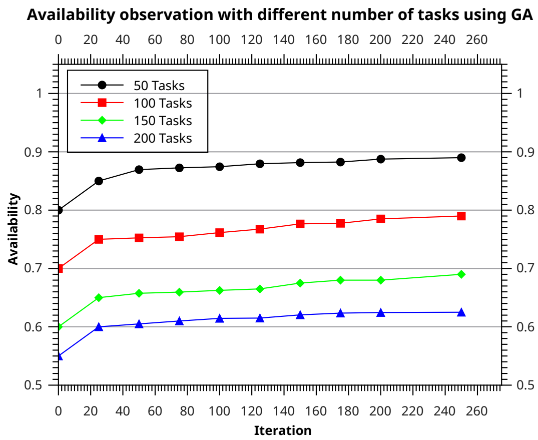

FIGURE 4.11: Availability observation with a different number of tasks using GA

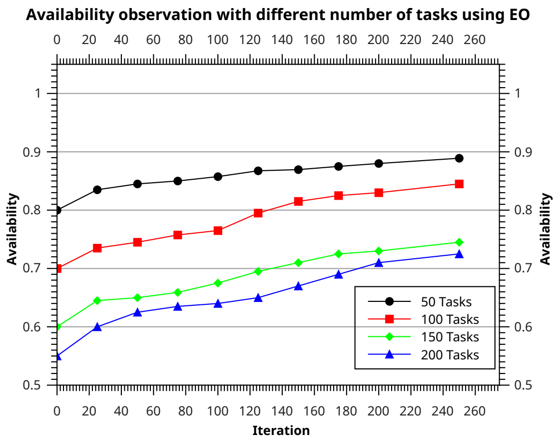

FIGURE 4.12: Availability observation with a different number of tasks using EO

4.3.5 Experiment with Varying Number of Nodes

We also study the effect of varying number of nodes on resource availability in the on-demand computing environment. For this experiment, we consider 8, 16, 24, and 32 nodes.

When the number of nodes increases, resource availability increases. It also shows that the MATS_ACO algorithm performs better than GA and EO algorithms. The mean and median values of resource availability are calculated as shown in TABLE 4.15.

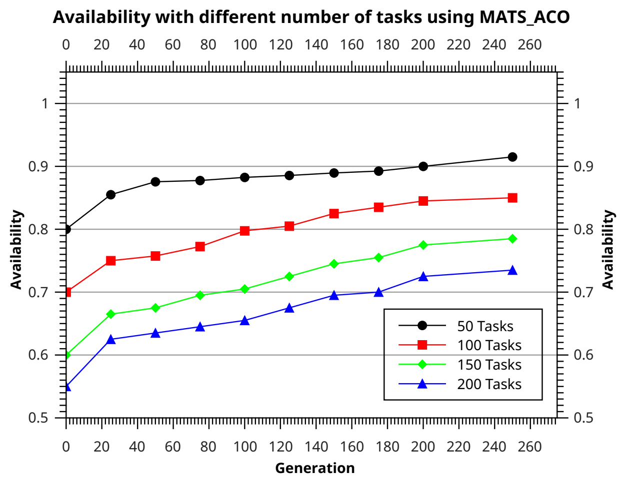

FIGURE 4.13: Availability observation with a different number of tasks using MATS_ACO

TABLE 4.15: The mean and median value of resource availability. FIGURE 4.14 shows the results for GA method while FIGURE 4.15 and FIGURE 4.16 shows the results for EO and MATS_ACO methods respectively.

Strategy	Number of nodes	mean	median
GA	8	0.70955	0.7175
	16	0.75965	0.762
	24	0.84214	0.8525
	32	0.9044	0.91
EO	8	0.73405	0.7803
	16	0.77127	0.7803
	24	0.8577	0.8675
	32	0.91405	0.9198
MATS_ACO	8	0.74345	0.7513
	16	0.8123	0.8125
	24	0.86345	0.8768
	32	0.92475	0.9328

4.3.6 Experiment with Varying Processing Speed of Nodes

In this experiment, we study the effect of speed of processing nodes on resource availability in the on-demand computing environment. We consider the number of nodes as 16. All the inpute are from TABLE 4.8.

When speed of the processing node increases, resource availability increases. Here also, our proposed algorithm outperforms GA and EO algorithms. The mean and median values

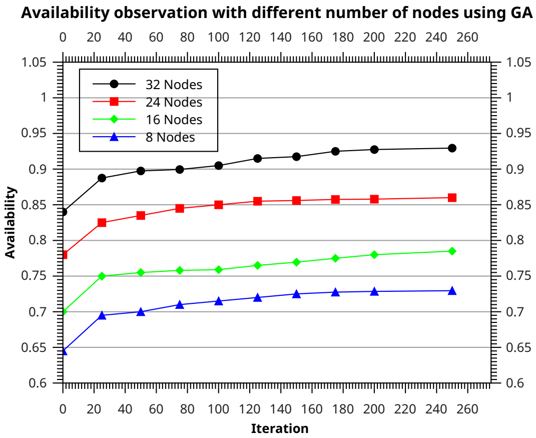

FIGURE 4.14: Availability observation with a different number of nodes using GA

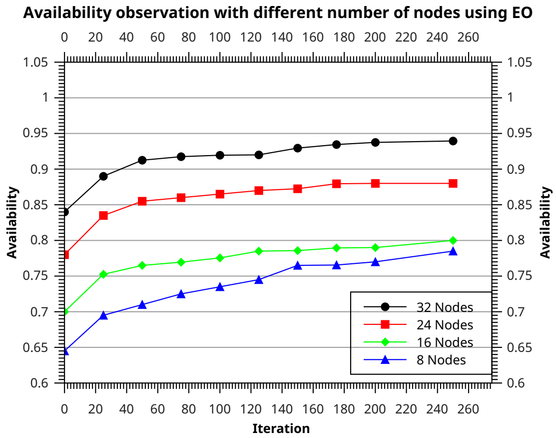

FIGURE 4.15: Availability observation with a different number of nodes using EO

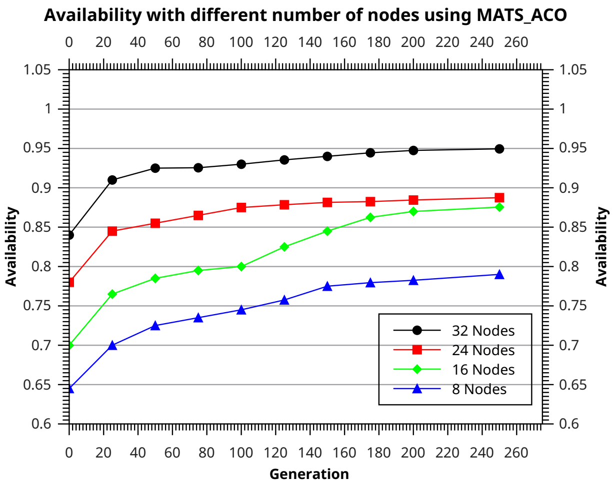

FIGURE 4.16: Availability observation with a different number of nodes using MATS_ACO

of resource availability are calculated as depicted in **TABLE 4.16**.

4.3.7 Experiment with Varying Load on Nodes

In this experiment, we study the effect of the load while calculating resource availability. For this experiment, we consider 16 nodes in the on-demand computing environment.

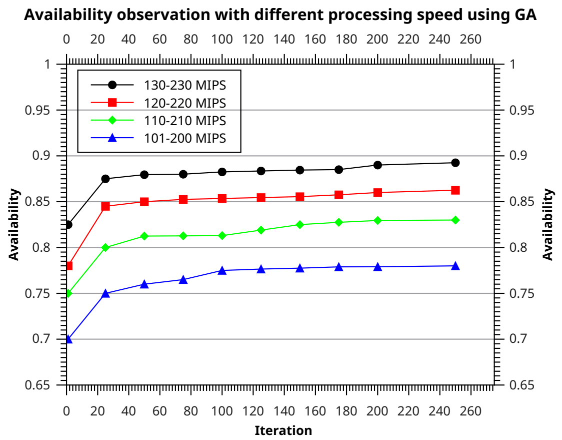

FIGURE 4.17: Availability observation with varying processing speed of nodes using GA

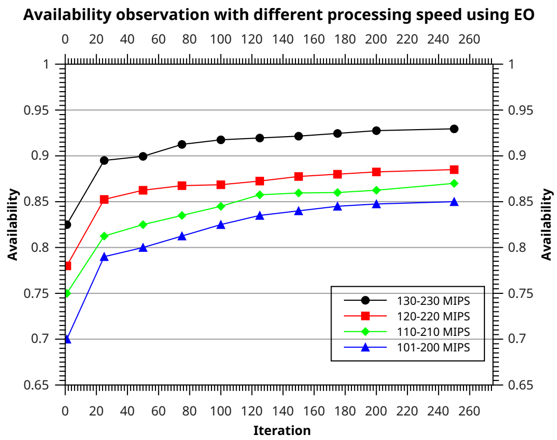

FIGURE 4.18: Availability observation with varying processing speed of nodes using EO

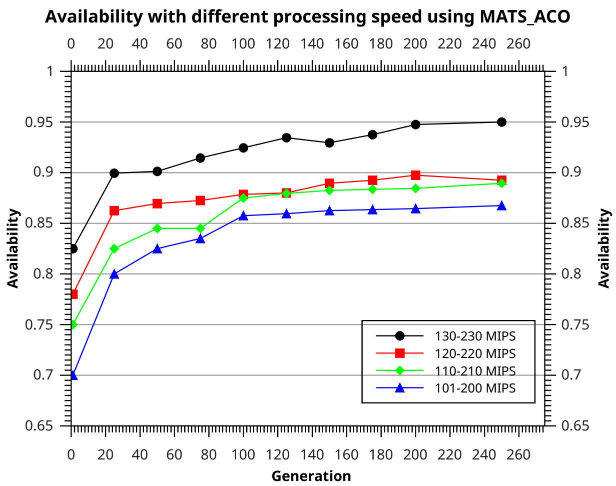

FIGURE 4.19: Availability observation with varying processing speed of nodes using MATS_ACO

When load on the node increases, resource availability decreases. We also see that our proposed algorithm works better than GA and EO algorithms. We also calculated the mean and median of resource availability results (as shown in TABLE 4.17).

TABLE 4.16: The mean and median value of resource availability. FIGURE 4.17 shows the results for GA method while FIGURE 4.18 and FIGURE 4.19 shows the results for EO and MATS_ACO methods respectively.

Strategy	Processing speed	mean	median
GA	101 – 200 MIPS	0.76419	0.7758
	110 – 210 MIPS	0.81192	0.816
	120 – 220 MIPS	0.8471	0.854
	130 – 230 MIPS	0.87775	0.883
EO	101 – 200 MIPS	0.8145	0.83
	110 – 210 MIPS	0.8377	0.8513
	120 – 220 MIPS	0.86285	0.8705
	130 – 230 MIPS	0.9072	0.9185
MATS_ACO	101 – 200 MIPS	0.8335	0.8585
	110 – 210 MIPS	0.85595	0.8773
	120 – 220 MIPS	0.8715	0.8793
	130 – 230 MIPS	0.916375	0.927

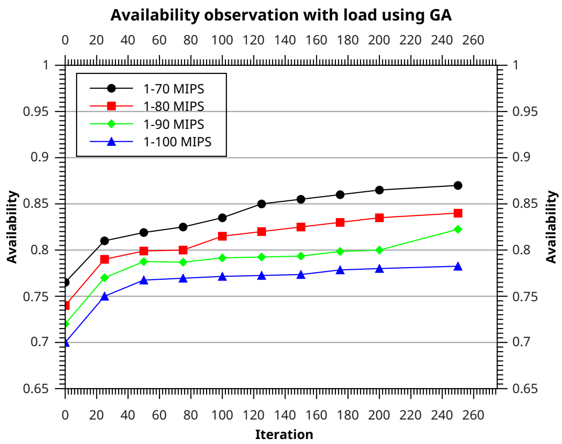

FIGURE 4.20: Availability observation with different loads in nodes using GA

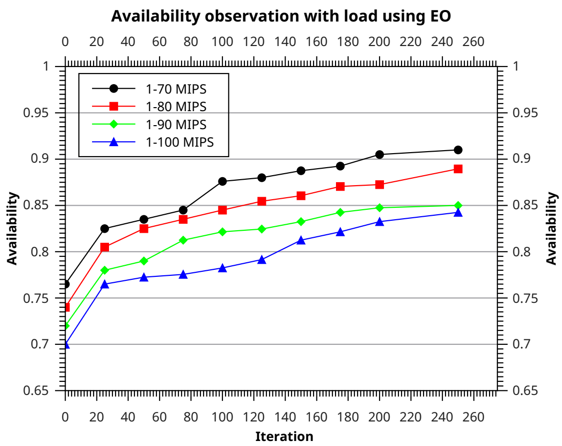

FIGURE 4.21: Availability observation with different loads in nodes using EO

4.3.8 Makespan Analysis

In this experiment, the makespan of the mentioned algorithms are evaluated and are compared. Since there may be several transactions in the environment of the on-demand computing, the simulations are done on the size and the number of the transactions. 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

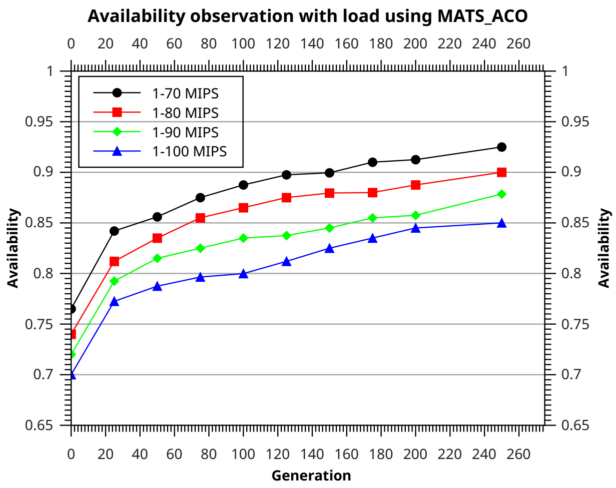

FIGURE 4.22: Availability observation with different loads in nodes using MATS_ACO

TABLE 4.17: The mean and median value of resource availability. FIGURE 4.20 shows the results for GA method while FIGURE 4.21 and FIGURE 4.22 shows the results for EO and MATS_ACO methods respectively.

Strategy	Load	mean	median
GA	1 – 70 MIPS	0.8354	0.8425
	1 – 80 MIPS	0.8094	0.8175
	1 – 90 MIPS	0.78629	0.792
	1 – 100 MIPS	0.76455	0.772
EO	1 – 70 MIPS	0.8621	0.878
	1 – 80 MIPS	0.83975	0.8498
	1 – 90 MIPS	0.8121	0.823
	1 – 100 MIPS	0.7896	0.787
MATS_ACO	1 – 70 MIPS	0.877	0.8925
	1 – 80 MIPS	0.8529	0.87
	1 – 90 MIPS	0.8261	0.8363
	1 – 100 MIPS	0.80234	0.806

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 three algorithms grows up as the number of the transactions or tasks increases.

We have the makespan analysis with varying number of tasks as shown in FIGURE 4.23, 4.24, 4.25 and 4.26. We see that our algorithm performs better than GA and EO. The mean and median of the makespan from the results depicted are calculated in TABLE 4.18.

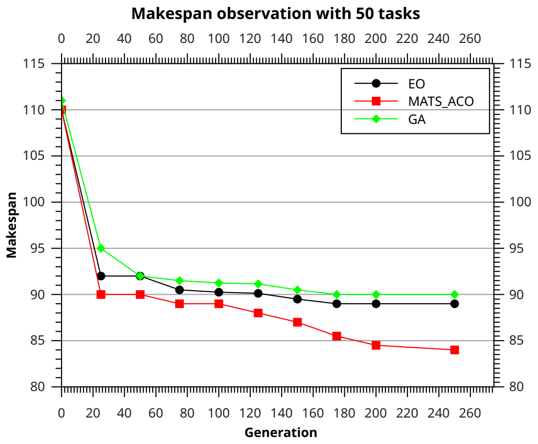

FIGURE 4.23: Makespan observation with different tasks with 50 tasks

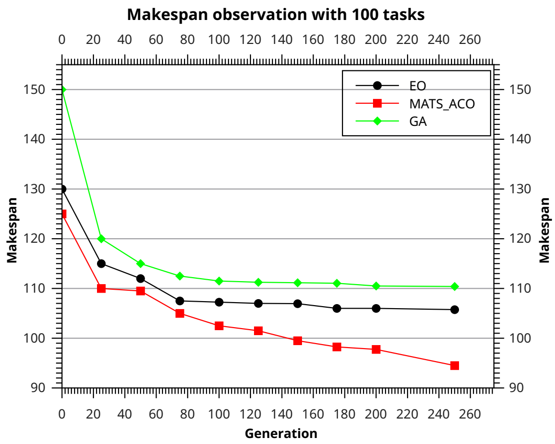

FIGURE 4.24: Makespan observation with different tasks with 100 tasks

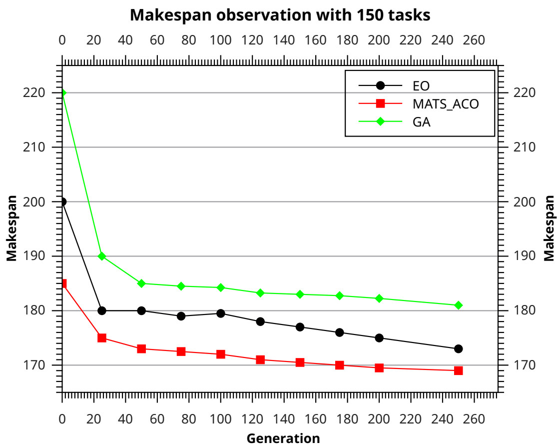

FIGURE 4.25: Makespan observation with different tasks: with 150 tasks

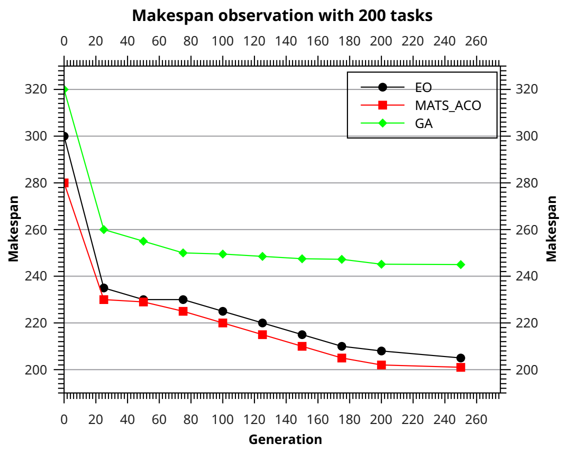

FIGURE 4.26: Makespan observation with different tasks: with 200 tasks

4.3.9 Comparative Study of Proposed Algorithm

We have the comparative analysis of proposed algorithm for availability analysis as shown in FIGURE 4.27 while in FIGURE 4.28 we see the comparative analysis of proposed algorithm for makespan analysis. We also calculated the availability based on time as shown in TABLE 4.20 while TABLE 4.19 shows the availability along with several iterations, i.e., 100, 200, and 300 in 40 simulations. In TABLE 4.19, we calculated average makespan with standard deviation and confidence interval (%95). In

TABLE 4.18: The mean and median value of makespan calculated from results in FIGURE 4.23, 4.24, 4.25 and 4.26

Number of Tasks	Strategy	mean	median
50	GA	93.24	91.2
	EO	92.1375	90.1875
	ACO	89.7	88.5
100	GA	116.335	111.375
	EO	110.345	107.125
	ACO	104.35	102
150	GA	187.6	187.6
	EO	179.75	178.5
	ACO	172.75	171.5
200	GA	256.79	249
	EO	227.8	222.5
	ACO	221.7	217.5

TABLE 4.19: Availability with 40 simulations

Strategy	Iteration	Average	Standard deviation	Confidence Interval (95%)	Max
MATS_ACO	300	0.9968	24.075389405	0.996 0.9979	0.9979
	200	0.9958	21.9933938945	0.9954 0.9958	0.996
	100	0.99999	23.7106384351	0.9999 0.99999	0.99999
EO	300	0.9932	26.4430243	0.993 0.994	0.994
	200	0.99	26.3075946	0.989 0.991	0.991
	100	0.9977	25.505675	0.997 0.998	0.998
GA	300	0.9882	27.545430987	0.9981 0.99825	0.99825
	200	0.989	26.935657525	0.989 0.9892	0.9892
	100	0.989	26.0567654	0.988 0.9899	0.9899

TABLE 4.20: Availability comparison of our proposed algorithm with EO and GA with respect to time

Time (time unit)	MATS_ACO	EO	GA
100	0.99999	0.9977	0.989
200	0.9958	0.99	0.9890
300	0.9968	0.9932	0.9882
400	0.9939	0.9928	0.9906
500	0.9911	0.9881	0.98
600	0.9911	0.9896	0.9885
700	0.9931	0.9905	0.9839
800	0.99	0.9887	0.9855
900	0.9894	0.9890	0.9842
1000	0.9895	0.9879	0.9879

TABLE 4.21 we see the makespan analysis based on number of tasks. In this table, we calculated makespan when number of tasks vary. We selected the minimum number of

TABLE 4.21: Makespan comparison of our proposed algorithm with EO and GA with respect to number of tasks

Number of Tasks	MATS_ACO	EO	GA
100	310	313	320
200	375	419	420
300	424	425	430
400	530	535	540
500	570	575	580
600	575	585	600
700	635	640	664
800	700	705	725
900	720	715	700
1000	720	725	735

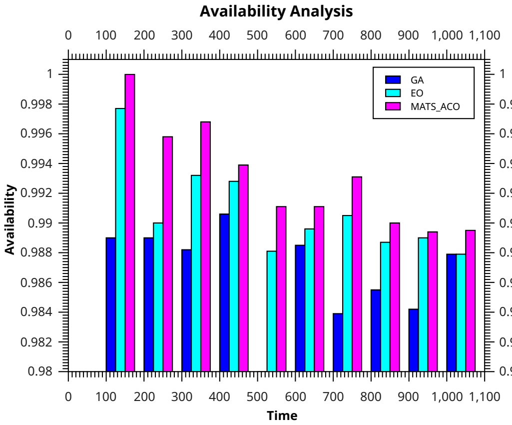

FIGURE 4.27: Availability Analysis

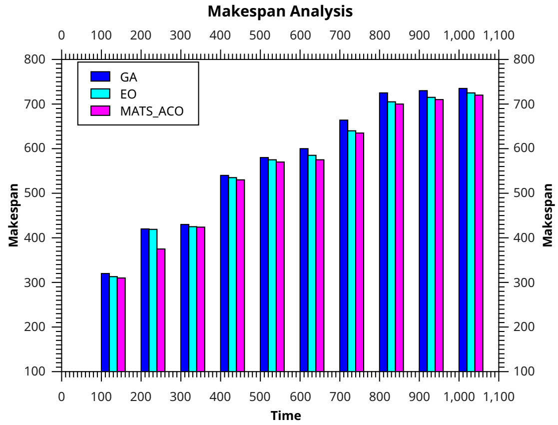

FIGURE 4.28: Makespan Analysis

TABLE 4.22: Makespan with 40 simulations

Strategy	Iteration	Average	Standard deviation	Confidence Interval (95%)	Max
MATS_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

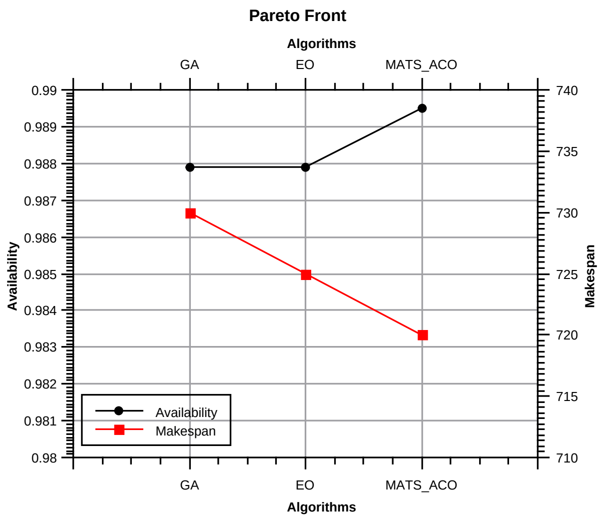

FIGURE 4.29: Analysis of the Pareto Front of the bi-objective optimization problem

tasks as 100 and maximum as 1000.

We have the pareto front analysis of the bi-objective problem in this chapter as shown in FIGURE 4.29. Here the algorithms are compared with both of availability and makespan in the same graph. In the figure, it is evident that availability increases and makespan decreases when we use MATS_ACO compared to GA and EO on an average value of time (we have chosen the time as $1000s$).

4.4 Summary

The maximization of the resource availability becomes one of the prime factors for transaction scheduling in on-demand computing system. The other objective is to minimize the makespan. In this chapter, we formulated the problem with multi-objective functions; maximizing availability and minimizing makespan. we have used ACO based transaction scheduling algorithm. We compared our proposed algorithm with two meta-heuristic scheduling algorithms based on EO and GA. The experimental results show that our proposed algorithm performs better than other two algorithms. We also carried out Wilcoxon statistical test for the validation of the results. The normality tests have been carried out using Shapiro-Wilk tests. For the network simulation we followed NFSNet scenario.

In this research we assumed independent transactions. For dependent transaction, the future research can further consider deadline constrained workflow scheduling approach.