Mansoura Journal of Computers and Information Sciences

Swarm Intelligence based Fault-Tolerant Real-Time Cloud Scheduler

A. S. Abohamama
Computer Science Dept.
University of Mansoura
abohamama@mans.edu.eg

M. F. Alrahmawy
Computer Science Dept.
University of Mansoura
mrahmawy@mans.edu.eg

Mohamed A. Elsoud
Computer Science Dept.
University of Mansoura
moh_soud@mans.edu.sa

Taher T. Hamza
Computer Science Dept.
University of Mansoura
Taher_hamza@yahoo.co

ABSTRACT
Cloud computing is a distributed computing paradigm that is deployed in many real-life applications. Many of these applications are real-time such as scientific computing, financial transactions, etc. Therefore, improving the dependability of cloud environments is extremely important to fulfill the reliability and availability requirements of different applications, especially real-time applications. Fault tolerance is the most common approach for improving the system's dependability. In addition to traditional fault tolerance techniques such as replication, job migration, software rejuvenation, etc, fault-tolerant scheduling algorithms can play a great role toward more dependable systems. In this paper, an ACO based fault-tolerant soft real-time cloud scheduler is developed to minimize deadlines missing rate, makespan, and the imbalance in distributing the workload among the different machines. The performance of proposed scheduler has been assessed under different scenarios. Also, it has been compared to other well-known scheduling algorithms and the experimental results have shown the superiority of the proposed algorithm.

General Terms
Artificial Intelligence, Swarm Intelligence

Keywords
Real-Time Applications, Ant Colony Optimization, Fault Tolerance, Cloud Scheduling

1. INTRODUCTION
Cloud computing is relatively recent computing paradigm that provides various services over the network on demand scalability [1]. This computing paradigm is based on a pay-as-you-go pricing model. Also, it reduces the up-front investment and maintenance costs [2]. Because of these advantages, the clouds have been adopted in many fields such as scientific research, e-commerce, health, etc [3]. Many applications that employ the clouds, such as scientific computation, financial transaction, and healthcare applications, are real-time and it demand specific reliability and availability requirements. The correct operation of these applications is not based only on the results correctness, but also on the time by which the results are generated [4].

On other side, in cloud based applications, most of computations are done on remote nodes which increase the probabilities of error occurrences due to the soft control over cloud nodes and unexpected network latency [5]. Also, many cloud providers uses inexpensive commodity hardware in building the cloud infrastructure which increases the probability of failure occurrence [4]. Hence, enhancing the cloud environments' dependability becomes an active research area, in industry and academia, because of its importance.

Fault Tolerance is the most common approach to build a dependable system in addition to some other approaches such as fault avoidance, and fault forecasting [6]. Fault tolerance means the system's ability to perform its function correctly despite of the occurrence of faults [7]. Many traditional techniques can be adopted to achieve the fault tolerance in the cloud environment such as preemptive migration, software rejuvenation, replication, and check-point/restart [8]. However, fault tolerant scheduling has proved its effectiveness in achieving the fault tolerance where the tasks are replicated and assigned to different computing nodes [4].

Using the virtualization technology, the computing resources of the cloud are usually provided dynamically on demand to the customers as an apparently unending group of interconnected virtual machines according to a group of service level agreements (SLA) established between the cloud providers and the customers [1]. Hence, the cloud consists of a large number of virtual machines (VMs). Also, it can execute many tasks and can offer many services for many clients in the same time. Therefore, assigning this huge number of tasks and services to the different VMs manually is a challenging task. So, an efficient scheduling algorithm, which is able to satisfy its design objectives, is required in cloud environments [9]. The cloud scheduler should be as efficient as possible because it can greatly affect the overall performance of the cloud system [10].

Task scheduling is an NP-complete problem where many heuristics and meta-heuristics have been employed by the researchers to solve it, seeking an optimal or near-optimal solution [9]. Swarm Intelligence is a category of bio-inspired algorithms which attempts to build meta-heuristics to solve complex problems by mimicking the collective behavior of swarms and their abilities in solving problems [11]. In this

paper, we use a well-known swarm intelligence technique called ant colony optimization (ACO) to develop a multi-objective fault tolerant soft real-time cloud scheduler to minimize the deadline missing rate, makespan and load imbalance degree (LIMD), while preserving the fault tolerance principle.

The remaining sections of the paper are organized as follow: Section 2 presents an overview on some research efforts which have been done to improve the fault tolerance of cloud environments through adopting the fault tolerant scheduling. Section 3 describes the main idea of the ant colony optimization algorithm. Section 4 introduces the problem statement and formulation. Section 5 presents a multi-objective ACO based fault-tolerant real-time cloud scheduler called FTRTS-MOACO. Finally, Sections 6 includes the conclusion and future work.

2. RELATED WORK

Generally, there are many approaches which have been developed to deal with the task scheduling problem in grid, multi-processors and distributed systems. However, these approaches are not suitable and cannot be applied directly in the cloud systems due to its different characteristics [10]. Therefore, the task scheduling problem in the cloud has attracted the researchers' attention and some attempts have been done to deal with this problem. But, using cloud environments for executing real-time applications is relatively new and few approaches have been developed to schedule the real-time tasks, while keeping their deadlines [5].

In [12], A particle swarm optimization (PSO) based scheduling algorithm is presented to schedule application workflows in the cloud environments. The proposed approach is concerned with minimizing execution costs of application workflows including transmission cost and computation cost. Another ant colony optimization (ACO) based workflow applications scheduling heuristic is proposed in [13]. The proposed algorithm called Load balancing optimization algorithm based on ant colony algorithm (ACO-LB). ACO-LB is concerned with enhancing the load balance in addition to minimizing the makespan. Also, in [14], bee swarm optimization based task scheduler that schedule tasks in the cloud resources with minimum makespan through an efficient workloads distribution. In [9], an ant colony optimization (ACO) based cloud task scheduler is proposed to schedule the cloud tasks on the hired virtual machines. The proposed scheduler is aiming at optimizing the total execution time or the makespan. Additionally, in [15], a number of well-known scheduling algorithms, including First Come First Serve (FCFS), Minimum Completion Time (MCT), Minimum Execution Time (MET), Max-min, Min-min and Sufferage, have been implemented to schedule independent tasks on cloud VMs. A number of experiments have been conducted and the performance of the different algorithms has been assessed using different performance measuring criteria.

In [16], a data locality driven cloud scheduling algorithm is proposed which is a fault tolerant version of Balance Reduce Algorithm (BAR). The proposed algorithm deals with the machine failures which occur during tasks execution. Also, like other data locality based scheduling algorithms, the proposed algorithm minimizes the network access, bandwidth usage, and makespan. Another fault tolerant cloud scheduling algorithm based on LCA optimization algorithm is proposed in [17]. It is called dynamic clustering league championship algorithm (DCLCA). The proposed algorithm aims at minimizing the makespan. Also, it employs task migration and fault detection strategies to reduce the task failure rate. Another lookahead genetic algorithm (LAGA) workflow applications scheduler is proposed in [18] which utilizes reliability-driven reputation to optimize reliability and makespan of distributed workflows. Another fault tolerant scheduling algorithm named MaxRe is proposed in [19] to achieve the desired reliability requirement for the user through determining the appropriate number of replicas, using a reliability analysis mechanism, for the different tasks.

In [10], a soft real-time scheduler is proposed based on particle swarm optimization (PSO) with optimized objectives: cost, makespan, deadline missing ratio, and load balance. In [20], they propose a multi-objective GA based scheduler to optimize energy consumption, gas emissions, and the total profit while taking into account the application's deadline. Another green computing supporting real-time energy-aware task scheduler named EARH is proposed in [21] which employs a rolling-horizon optimization policy. In [22], they propose a real-time utility accrual scheduler where tasks are scheduled in a non-preemptive manner to optimize the total utility using profit and penalty utility functions. In [23], a near-optimal computationally efficient greedy real-time task scheduler is proposed for scheduling real-time batch jobs with objectives is to maximize the social welfare and revenues using parallelism. In [24], a reliability-aware scheduler which uses a reliability assessment model is proposed based on the node's reputation in cloud infrastructure; it is used for general as well as real-time applications. Finally, two cloud schedulers have been proposed in [25] based on greedy algorithm and adaptive genetic algorithm. The proposed schedulers have been designed to schedule hard real-time tasks with precedence on heterogeneous VMs.

3. ANT COLONY OPTIMIZATION

Ant colony optimization (ACO) is an optimization algorithm which proposed by M. Dorigo in the early 1990s. It is inspired by the foraging behaviour of real ant colonies. To find food sources, ants leave their nest and take random paths to scan the surrounding area. While moving, the ants put smelly trails called pheromone on the paths. When choosing a path, the ants prefer to choose the path that has the strongest pheromone concentration. When an ant finds a food source, it takes some of it and leaves pheromone on the path during its return to the nest. With the repetition of this process, the shortest paths get high pheromone concentrations. These paths attract the ants to follow during their next trips [9]. In this insect society, ants communicate indirectly through modifying the environment to support cooperation among themselves toward their target. This type of communication is called stigmergy [1, 26, 27].

ACO is a metaheuristic for solving combinatorial optimization problems through mimicking the behavior of the real ants [28]. Practically, each artificial ant attempts to build a solution for the optimization problem under concern. Each ant leaves pheromone trails on the path it takes to reach this solution. The pheromone concentration is proportional to the quality or fitness of the solution. Next ants then attempt to build their own solutions, but they are affected by the pheromone trails left on the paths by their predecessors [28]. ACO has pros and cons. The pros of ACO include the use of the positive feedback mechanism, inner parallelism and extensible. The cons of ACO metaheuristic include overhead and the stagnation phenomenon [9]. ACO algorithm has a set of main steps including pheromone initialization, a candidate solution construction by every single ant, pheromone update process (evaporation, intensification). The flowchart of the ACO algorithm is shown in Figure 1 [28].

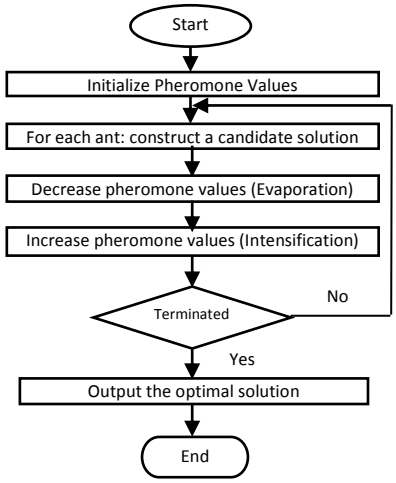


Fig 1: Flowchart of the standard ACO

ACO has been successfully used to solve many optimization problems such as the traveling salesman problem [29], the flow shop scheduling problem [30], the quadratic assignment problem [31], and task scheduling [9]. In this paper, ACO has been used to a build fault-tolerant soft real-time cloud scheduler which is described below in detail.

4. PROBLEM STATEMENT AND FORMULATION

Computing resources in virtualized clouds are provisioned in the form of virtual machines (VMs) that can run separately and independently on the same physical machine or on different ones to execute assigned tasks. In this paper, the task is the smallest identifiable piece of work that achieves a specific service/function and it is the smallest schedulable entity. A real-time task is no more than an ordinary task with a deadline. Given a set of real-time tasks, $\boldsymbol{T} = \{t_1, t_2, t_3 \dots, t_n\}$ where each $t_i \in \boldsymbol{T}$ has a group of attributes (ac_i, w_i, e_i, f_i, and d_i) which represent arrival, waiting, expected execution, expected finish and deadline times of t_i, respectively. Also, given a set of virtual machines, $\boldsymbol{VM} = \{vm_1, vm_2, vm_3, \dots, vm_M\}$ that represents the hired computing power. Then, the role of cloud scheduler is to assign the real-time tasks to the different VMs in a way that achieve its design objectives which include minimizing deadlines missing and makespan in addition to achieving good balance in distributing the workloads among the different VMs.

The replication mechanism is used in our scheduler to achieve the fault tolerance, where each t_i has *nr* replicas. The expected finish time of each replica is computed by Eq. 1.

$$f_{k,r,j} = ac_k + w_{k,r} + e_{k,r,j} \quad (1)$$

where *k* is the index of the original real-time task, *r* is the index of the current replica and *j* is the index of allocated virtual machine index, ac_k is the arrival time of original tasks at the cloud, $w_{k,r}$ is the waiting time of replica *r* on VM_j, and $e_{k,r,j}$ is the expected execution time of the replica on VM_j which can be computed by Eq. 2.

$$e_{k,r,j} = \frac{TL_k}{Pe\_Num_j * Pe\_Mips_j} \quad (2)$$

where TL_k is the length of the task t_k in million instructions (MI), Pe_Num_j is the processing elements number of VM_j, and Pe_Mips_j is the computing power of each processing element in VM_j represented by Million Instructions per Second (MIPS).

After executing the different replica of real-time task t_i, the status of t_i is "*Passed*" if one of its replicas, at least, finished execution before d_i. Otherwise, the status is "*Failed*". In the proposed work, *Fail_Num* represents the gross number of failed real-time tasks.

The total time needed by a certain virtual machine VM_j to finish a number of assigned tasks NT_j is denoted by *CTime(VMj)* and can be computed by Eq.3

$$CTime\left(VM_j\right) = \frac{\sum_{t=1}^{NT_j} TL_i}{Pe\_Num_j * Pe\_Mips_j} + \frac{\sum_{t=1}^{NT_j} IFS_i}{BW_j} \quad (3)$$

where TLi is the length of task t_i in MI, IFSi is the size of the input file of task t_i, Pe_Num_j* Pe_Mips_j is the total computing power of VM_j, and BW_j is the bandwidth of VM_j. In the context of task scheduling, *Makespan* of a set of tasks is the total time that elapses from the start time to the finish time and can be computed by Eq. 4 [15].

$$Makespan = \max\left(CTime\left(VM_j\right)\right),\ where\ 1 \le j \le M \quad (4)$$

where *M* is the total number of hired VMs.

Also, load balancing is a major issue in designing any task scheduler. In the proposed work, the term "Load Imbalance Degree" or *LIMD* is used to represent the imbalance in distributing the workload among the different VMs. Therefore, the smaller the value of *LIMD* is, the better the performance of the scheduler in terms of resource utilization. *LIMD* is computed by Eq.5 [15].

$$LIMD = \frac{Makespan - MinCTime}{AvgCTime} \quad (5)$$

where *Makespan*, *MinCTime* and *AvgCTime* are computed by Equations 4, 6, and 7, respectively.

$$MinCTime = Min\left(CTime\left(VM_j\right)\right), where\ 1 \le j \le M \quad (6)$$

$$AvgCTime = \frac{\sum_{j=1}^{M} CTime\left(VM_j\right)}{M} \quad (7)$$

From the aforementioned description, the task scheduling problem can be defined as a multi-objective optimization problem as shown in Eq. 8.

$$\text{Minimize } (Makespan, LIMD, Fail\_Num) \quad (8)$$
$$S.T. \forall\ t_i \in T, w_i + e_i \le d_i (if\ the\ status\ of\ t_i = passed)$$

5. PROPOSED ALGORITHM (FTRTS-MOACO)

A new scheduling algorithm is proposed here, with a main objective is to fulfill the fault tolerance for real-time tasks that are running on cloud environment. The name of the proposed algorithm is **F**ault **T**olerant **R**eal-time **T**ask **S**cheduling based on **M**ulti-**O**bjective **A**nt **C**olony **O**ptimization (**FTRTS-MOACO**). The pseudocode of **FTRTS-MOACO** is shown in Algorithm1. **FTRTS-MOACO** starts by applying the replication process to achieve the fault tolerance. It generates a real-time task list, ***RT-List***$= \{t_{1,1}, t_{1,2}, t_{1,2}, t_{2,1}, t_{2,2}, t_{2,3} \dots, t_{n,1}, t_{n,2}, t_{n,3}\}$, by tripling the original list *RT-List*$= \{t_1, t_2, t_3 \dots, t_n\}$, *i.e., nr* =3, where $t_{k,1}$, $t_{k,2}$ and $t_{k,3}$ are the replicas of the original real-time task t_k. For more simplicity, an easier indexing mechanism is used in the rest of this paper instead of this doubly indexing in which $t_{k,r}$is replaced by t_i where $i = nr(k-1) + r = 3(k-1) + r$. After applying that, the replicated list becomes ***RT-List*** $= \{t_1, t_2, t_3 \dots, t_{N-2}, t_{N-1}, t_N\}$ where $N = nr(n-1) + nr = n \times nr$.

Algorithm1: Multi-objective ACO Scheduler

Input:

- List of n real-time tasks, RT-List
- List of M virtual machines , VM-List
- Number of ants, AntsNum
- Number of iterations, I_{max}

Output: optimal schedule, **S**

1. Apply the tripling process on the original RT-List to generate TRT-List.
2. Create and initialize the pheromone matrix, τ of size N × M, using Eq. 9.
3. *OptimalFitness* = $+\infty$.
4. **For** i=1,2,……, AntsNum
5. Create an ant and put it in the ants list, AntList.
6. **For** i=1,2,……, I_{max}
7. **Foreach** ant ∈ AntList **do**
8. *Schedule* = *MakeTour* (*TRT-List*, *VMList*). /* (Algo. 2)*/
9. Set *ScheduleFitness* for *Schedule* using Eq. 10.
10. **If** (*ScheduleFitness* < *OptimalFitness*)
    S = *Schedule*.
    OptimalFitness = *ScheduleFitness*.
11. Evaporate the pheromone matrix, *τ,* using Eq. 12.
12. **Foreach** ant ∈ AntList **do**
13. Intensify the pheromone matrix, *τ,* locally using Eq. 13.
14. Intensify the pheromone matrix, *τ,* globally using Eq. 15.
15. **Return S**

In order to use the ACO algorithm, it is important to define the pheromone information in such a way that reflects the most paramount information for solution construction. In the proposed algorithm, the pheromone information is encoded as pheromone matrix,$\boldsymbol{\tau}$ of size $N \times M$. The pheromone value $\boldsymbol{\tau}[i,j]$represents the desirability to assign t_i to VM_j where $i \in [1,N]$ and$j \in [1,M]$. τ is initialized using Eq. 9 [9].

$$\boldsymbol{\tau}[i,j] = c, \qquad where\ c > 0 \tag{9}$$

where *c* is a user-defined small positive value.

During each iteration, each ant makes a tour to create a possible schedule using Algorithm 2. Then, the fitness value of the proposed schedule is computed using Eq. 10.

$$f = \gamma * Fail\_Num\delta * LIMD + \zeta * (\frac{Makespan}{SingularExecutionTime}) \tag{10}$$

Where γ, δ and ζ are weighting parameters with total summation equal to 1. *SingularExecutionTime* is put as a denominator in the third term to reduce the effect of *MakeSpan* which has a range of values larger than the ranges of the first and second terms, i.e., *SingularExecutionTime* is used to normalize *MakeSpan*. It can be computed using Eq. 11.

$$SingularExecutionTime = \frac{\sum_{i=1}^{N} TL_i}{\sum_{j=1}^{M} Pe\_Num_j * Pe\_Mips_j} \tag{11}$$

In the evaporation process, all pheromone values in the pheromone matrix,τ, are decreased by a fixed proportion, ρ, at the end of each iteration using Eq. 12 [9].

$$\boldsymbol{\tau}[i,j] = (1-\rho) * \boldsymbol{\tau}[i,j], \qquad where\ 0 < \rho < 1 \tag{12}$$

The pheromone values are intensified at two levels: local and global. In the first level, each ant updates the pheromone values proportional to the quality of its candidate solution using Eq. 13 [9].

$$\boldsymbol{\tau}[i,j] = \boldsymbol{\tau}[i,j] + \Delta * \boldsymbol{TM}[i,j] \tag{13}$$

where Δ is the pheromone update value which is computed using Eq. 14 and $\boldsymbol{TM}$is the binary tour matrix.

$$\Delta = \frac{\lambda}{f_t^k} \tag{14}$$

where f_t^k is the fitness value of the tour made by ant *k* at iteration *t*. in the second level, the pheromone matrix,τ, is intensified only by the best ant, which produced the best tour during the iteration, using Eq. 15.

$$\boldsymbol{\tau}[i,j] = \boldsymbol{\tau}[i,j] + \frac{\lambda * \boldsymbol{TM}[i,j]}{f_t^*} \tag{15}$$

where f_t^* is the fitness value of the best tour generated during the current iteration.

Algorithm2: MakeTour Operation

Input:

- List of N real-time tasks, RT-List
- List of M virtual machines , VM-List

Output: optimal schedule, **S**

1. Create a zero matrix TM of size N×M.
2. Create Probabilistic Transition Matrix, *P* of size N×M.
3. Initialize P using Eq. 16.
4. **Foreach** *t* ∈ *TRT-List* **do**
5. Set *TaskId* to the index of *t.*
6. Create a list L, a list of VMs which cannot be allocated to replica t.
7. *Allowed* ← {Vm-List – L}.
8. Update *P* using Eq. 18.
9. Select VM ∈ *Allowed* using Roulette Wheel Selection.
10. Set *VmId* to the index of selected VM.
11. *TM [TaskId, VmId] =1.*
12. **Return** *TM*

Algorithm 2 is employed by each ant to make a tour to generate a possible schedule. It starts by creating a binary $N \times M$ tour matrix,$\boldsymbol{TM}$, which represents the schedule matrix. Then, another $N \times M$ probabilistic transition matrix $\boldsymbol{P}$ is created and initialized using Eq. 16 [9]. $\boldsymbol{P}$ includes the probabilities of choosing a specific VM to execute a certain real-time replica t_i. For example, if $\boldsymbol{P}$ [4,5] = 0.3, then VM_5 can be chosen for executing the replica t_4 with a probability value of 0.3 [9].

$$\boldsymbol{P}[i,j] = \frac{[\tau_{ij}(0)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{s=1}^{M} [\tau_{is}(0)]^{\alpha} * [\eta_{is}]^{\beta}} \tag{16}$$

where $\boldsymbol{\tau}_{ij}$ is the pheromone value of allocating Vm_j to the replica t_i at iteration zero, $\boldsymbol{\eta}_{ij}$ is an additional problem-dependent heuristic parameter that helps the ant in selecting Vm_j where t_i causes the smallest load value, and α and β are used to determine the relative weight of pheromone value and heuristic value, respectively. The value of $\eta_{ij}$is computed using Eq. (17).

$$\boldsymbol{\eta}_{ij} = \frac{1}{CTime_{ij}} \tag{17}$$

where $CTime_{ij}$ represent the time needed to execute the replica t_i if VM_j is allocated to it. It is computed using Eq. (18) [9].

$$CTime_{ij} = \frac{TL_i}{Pe\_Num_j * Pe\_Mips_j} + \frac{IFS_i}{BW_j} \tag{18}$$

where TL_i is the length of t_i (MI), IFS_i is the input file size of t_i, Pe_Num_j is the number of processing elements of Vm_j, Pe_Mips_j is the computing power of the processing elements in Vm_j in (MIPS), and BW_j is The bandwidth of Vm_j.

The replicas of a certain task are allocated to different VMs to satisfy the fault tolerance requirement. Hence, before selecting a VM to execute a real-time replica t_i, a list of Not-allowed VMs, *L*, is created for the current replica. This list contains the VMs previously allocated to other replicas of the same real-time task. Then, the list of allowed, *Allowed*, VMs is specified. After that, $\boldsymbol{P}$ is updated using Eq. 19. The update process considers the current replica to be assigned and the previously assigned replicas for each VM.

$$\boldsymbol{P}[i,j] = \begin{cases} \frac{[\tau_{ij}]^{\alpha} * [\eta_{ij}^{total}]^{\beta}}{\sum_{s \in Allowed}^{M} [\tau_{is}]^{\alpha} * [\eta_{ij}^{total}]^{\beta}}, if\ vm_j \in Allowed \\ 0, \qquad otherwise \end{cases} \quad (19)$$

where

$$\eta_{ij}^{total} = \frac{1}{TotalCTime\ (VM_j)} \quad (20)$$

and

$$TotalCTime\ (VM_j) = \quad CTime_{ij} + \sum_{w\ \epsilon\ real-time\ replicas\ already\ assigned\ to\ VM_j} CTime_{wj} \quad (21)$$

After updating ***P***, a VM is chosen by employing the roulette wheel selection algorithm, and the tour matrix ***TM*** is updated. Finally, after allocating a VM for each real-time replica, the tour matrix ***TM*** is returned.

6. Implementation and Evaluation

The performance of **FTRTS-MOACO** is assessed using a custom-made stimulated cloud environment built using c#. The experiments are conducted on two datacenters which contains 15-45 virtual machines. Table 1 shows the simulation parameters.

Table1: the default values of simulation parameters

Entity	Parameter	Value
Real-Time Task	Task length	1000-30000 MI
	Size of input file	300 Byte
	Deadline Time	25-175 Second
	Number of Replicas	3
Virtual Machines	MIPS	1000-4000
	RAM	256-2048
	Bandwidth	1000
	Number of PEs	1
Data Center	Number of Data Centers	2
	Number of Hosts	3-6

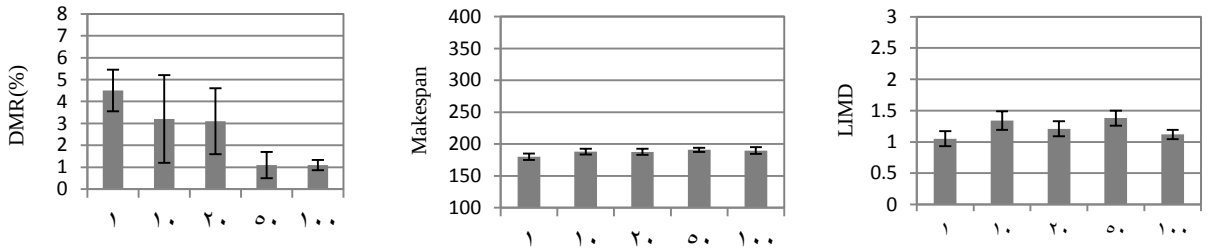
Four groups of experiments are conducted to assess the performance of FTRTS-MOACO. The first group evaluates its performance by applying a number of values for each parameter. The other groups compare the performance of FTRTS-MOACO to three other fault-tolerant real-time task scheduling algorithms. The first algorithm (FTRTS-ACO) is a variation of the algorithm introduced in [9], which is based on ant colony optimization. The second and third algorithms are FTRTS-MIN and FTRTS-MAX, which we implemented as extended versions of the well-known task scheduling heuristic algorithms Min-Min and Max-Min [15][30]. In our implementation of these three algorithms, we added replicas of the scheduled tasks and assigned deadlines to them. The performance of FTRTS-MOACO is compared to other scheduling algorithms in terms of makespan, LIMD, fitness, and deadline missing rate (DMR) that is computed by Eq. 22.

$$DMR = \frac{Fail\_Num}{n} \quad (21)$$

where *n* is the number of real-time tasks and *Fail_Num* represents the number of failed real-time tasks. For FTRTS-MOACO and FTRTS-ACO, the results are the average of ten executions. Additionally, 95% confidence interval (CI) of each result is shown in the graphs.

6.1 FTRTS-MOACO Evaluation

FTRTS-MOACO used a rich set of parameters which includes α, β, ρ, λ, γ, δ, ζ, *AntsNum*, and I_{max}. Different values of these parameters are used to address the effect of each one on the performance of proposed algorithm. The default values are (α =1, β =1, ρ =.5, λ =10, γ =.5, δ =.3, ζ =.2, AntsNum =10, and MaxIteration =10). The experiments are conducted using 200 real-time tasks with deadlines [50-150] and 30 VMs. The set of values are: *AntsNum* ∈ (1, 10, 20, 50, 100), I_{max} ∈ (10, 20, 50, 100, 500, 1000), α ∈ (0.1, 0.2, 0.3, 0.6, 1), β ∈ (0, 0.5, 1, 2, 3), ρ ∈ (0, 0.2, 0.5, 0.7, 0.9), and λ ∈ (1, 10, 100, and 500). The results of these experiments are shown in Figures 2-7, respectively.

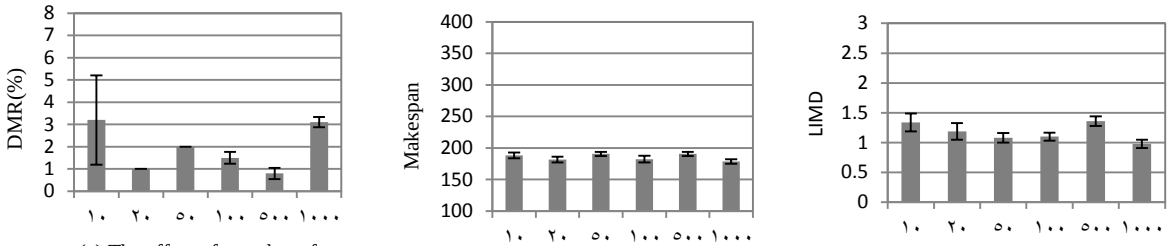


(a) The effect of number of ants on DMR

(b) The effect of number of ants on Makespan

(c) The effect of number of ants on LIMD

Fig 2: Performance of FTRTS-MOACO using different numbers of ants

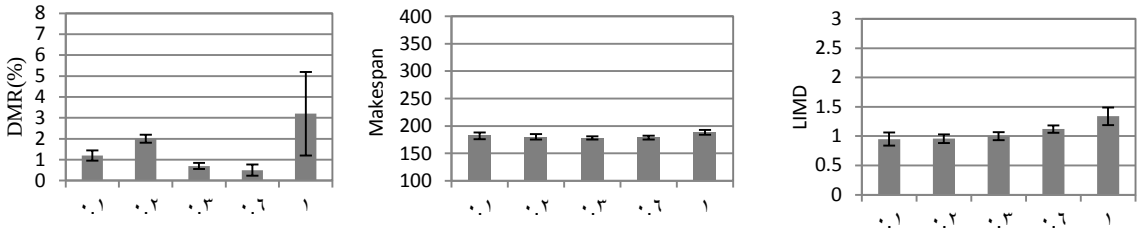


(a) The effect of number of iterations on DMR

(b) The effect of number of iterations on Makespan

(c) The effect of number of iteration on LIMD

Fig 3: Performance of FTRTS-MOACO using different iterations numbers

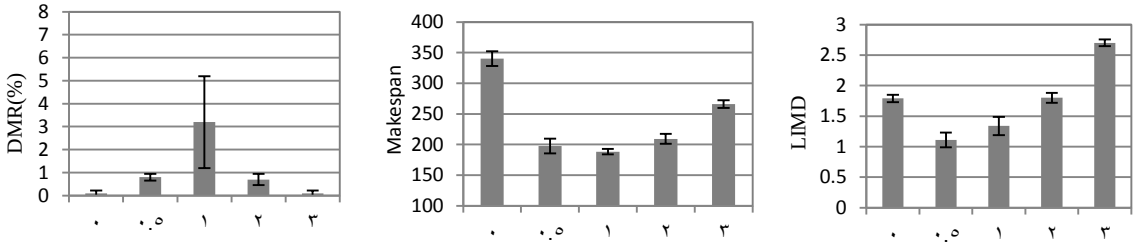


(a) The effect of α on DMR

(b) The effect α on Makespan

(c) The effect of α on LIMD

Fig 4: Performance of FTRTS-MOACO using different values for the parameter α

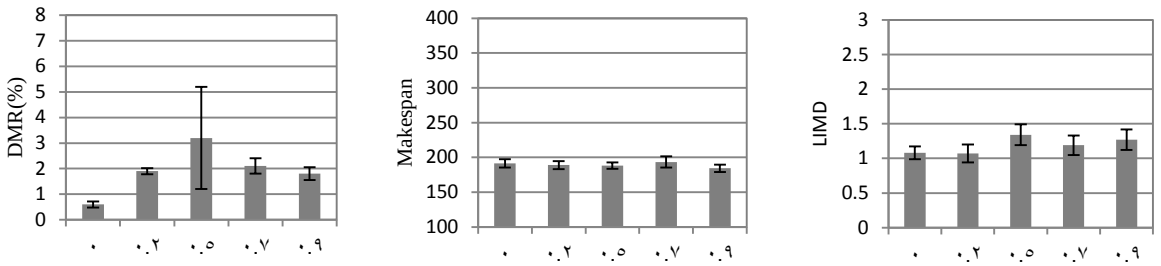


(a) The effect of β on DMR

(b) The effect of β on Makespan

(c) The effect of β on LIMD

Fig 5: Performance of FTRTS-MOACO using different values for the parameter β

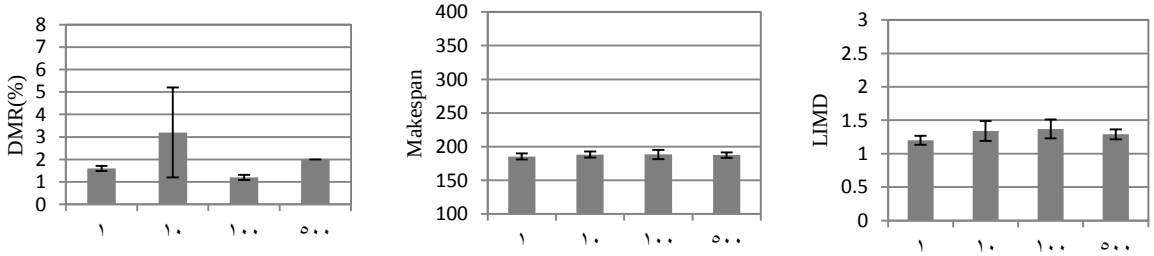


(a) The effect of ρ on DMR

(b) The effect of ρ on Makespan

(a) The effect of ρ on LIMD

Fig 6: Performance of FTRTS-MOACO using different values for the parameter ρ



(a) The effect of λ on DMR

(b) The effect of λ s on Makespan

(c) The effect of λ on LIMD

Fig 7: Performance of FTRTS-MOACO using different values for the parameter λ

The conducted experiments have shown the effect of each parameter on the different performance measures of the proposed scheduler. The best values of each parameter regarding the different performance measures are shown in Table 2.

From Table 2, we can observe that a single ant is sufficient to find an optimized solution that minimizes the makespan or minimizes the load imbalance because the behavior of many ants would be the same because of using the probabilistic transition matrix as road map toward the optimum solution, where the solution space itself is narrowed due to inhibiting the assignment of replicas of the same task to the same virtual machine. While a single ant is not sufficient, as demonstrated by the first group, to a find an optimized solution that minimizes the deadline missing ratio because adding the deadline constraints makes the optimization problem is much harder and requires more ants to cooperate in finding an optimum solution. In order to provide a balanced performance regarding the different performance measuring criteria, the following parameter values are used in the remaining experiments: AntsNum =100, I_{max} =100, α =0.6, β=1, ρ=0.2 and λ=1.

Table 2: The best values for the different parameters

Criteria	*AntsNum*	I_{max}	α	β	ρ	λ
DMR	100	500	0.6	3	0	100
Makespan	1	1000	0.3	1	0.9	1
LIMD	1	1000	0.1	0.5	0.2	1

6.2 Evaluating the Different Scheduling Algorithms Using Different Deadline Ranges

In this section, the performance of FTRTS-MOACO is evaluated against FTRTS-ACO, FTRTS-MIN, and FTRTS-MAX using 200 real-time tasks of different deadline ranges namely, [25-125], [50-150], and [75-175] with 30 VMs. The obtained results are appeared in Figure 8.

Based on Figure 8, we can see that the DMR of different scheduling algorithms is clearly decreased with enlarging the deadline ranges. This notice is justified, where the task of large deadline has a higher probability to meets its deadline. However, FTRTS-MOACO provides the best DMR followed by FTRTS-ACO then FTRTS-MIN and finally FTRTS-MAX. On the other side, regarding Makespan and LIMD, FTRTS-MAX provides the best values followed by FTRTS-MIN then FTRTS-MOACO and finally FTRTS-ACO. FTRTS-MOACO provides the best fitness values achieving a good balance between the different scheduling objectives.

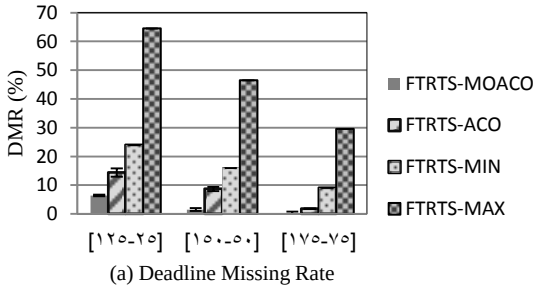
6.3 Evaluating the Different Scheduling Algorithms Using Different Number of Tasks

In this section, the performance of FTRTS-MOACO is evaluated against FTRTS-ACO, FTRTS-MIN, and FTRTS-MAX using different numbers of tasks namely, 50, 100, 150 and 200 with deadline range of [50-150], and 30 VMs. The obtained results are appeared in Figure 9.

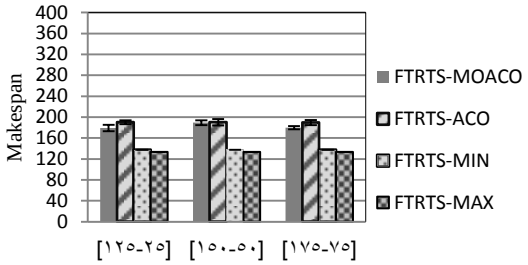
Based on Figure 9, we can observe that the different performance measures are influenced by increasing the number of tasks, particularly, DMR and fitness values with FTRTS-MAX. Once again, FTRTS-MOACO provides the best DMR and Fitness values achieving the highest degree of fault tolerance and the highest degree of balance among the different scheduling objective. On the other side, FTRTS-MAX and FTRTS-MIN, in order, are providing the best values regarding Makespan and LIMD. FTRTS-ACO performs well on the different criteria but FTRTS-MOACO is still better

6.4 Evaluating the Different Scheduling Algorithms Using Different Number of VMs

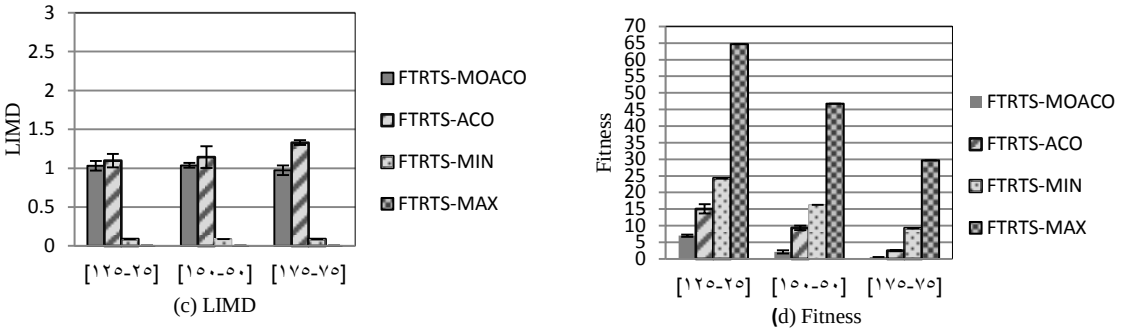
In this section, the performance of FTRTS-MOACO is evaluated against FTRTS-ACO, FTRTS-MIN, and FTRTS-MAX using 200 real-time tasks of deadline range [50-150] executed on a different number of VMS namely, 15, 30, and 45. The obtained results are shown in Figure 10



(a) Deadline Missing Rate

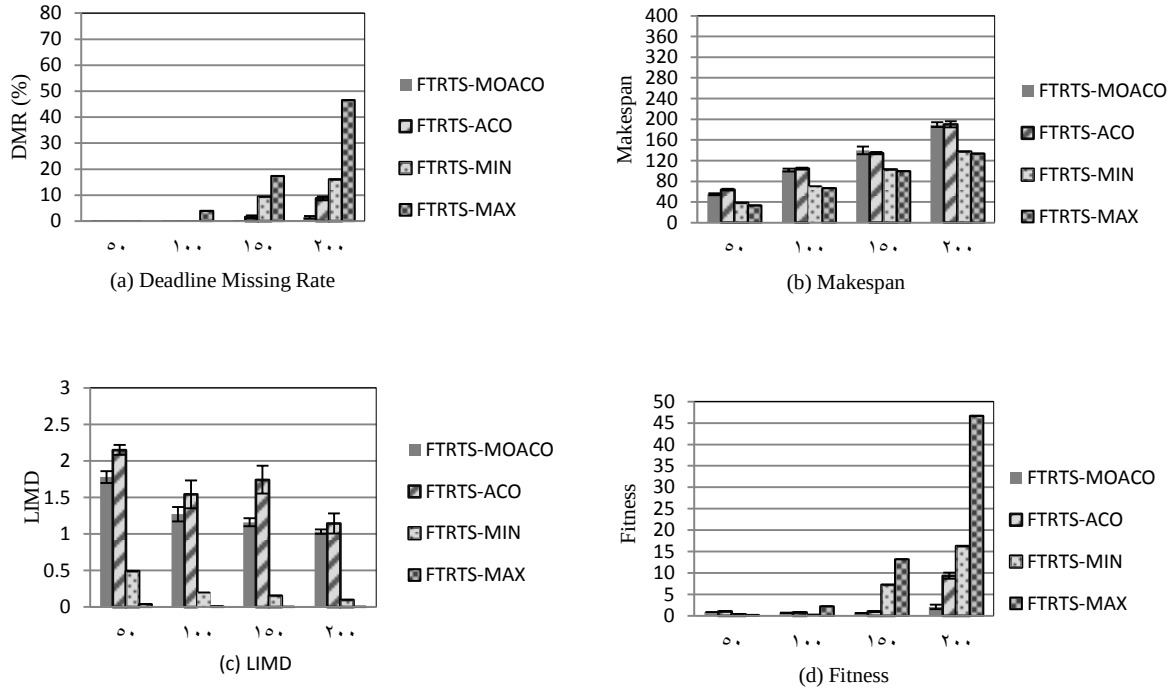


(b) Makespan



(c) LIMD (d) Fitness

Fig 8: Performance of different scheduling algorithms using different deadline ranges

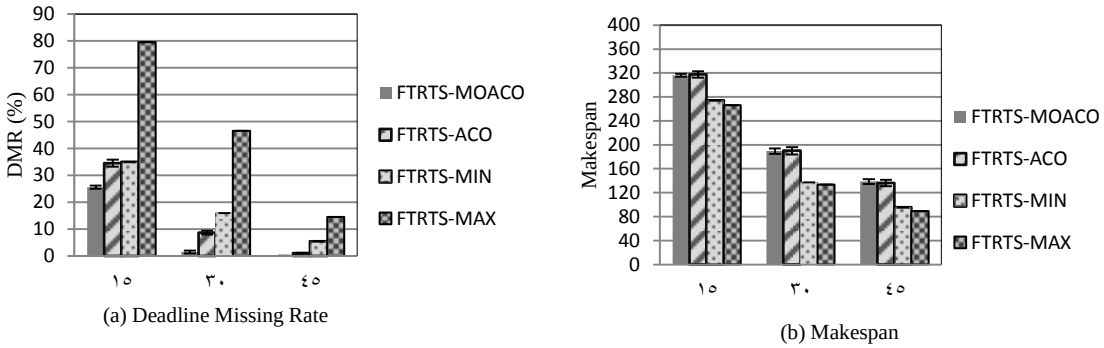


(a) Deadline Missing Rate (b) Makespan

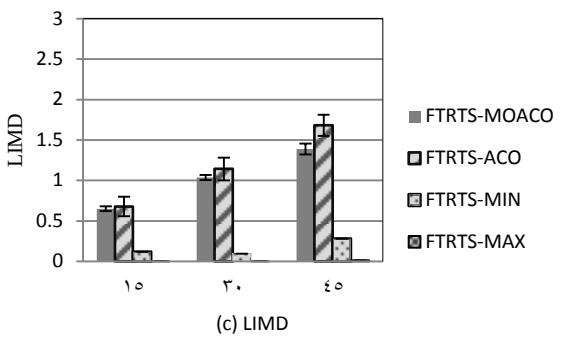
(c) LIMD (d) Fitness

Fig 9: Performance of the different scheduling algorithms using different number of tasks

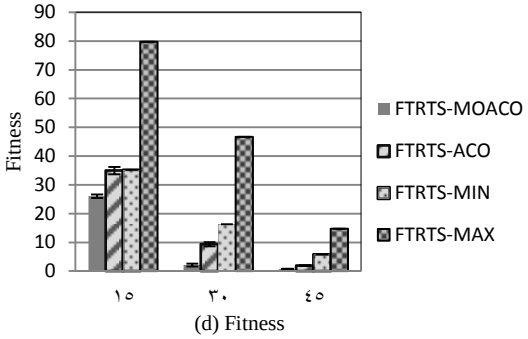
.



(a) Deadline Missing Rate (b) Makespan



(c) LIMD



(d) Fitness

Fig 10: Performance of the different scheduling algorithms using different number VMs

Based on Figure 10, we can notice that the different performance measures are greatly improved by increasing the number of used VMS except LIMD. Once again, we can notice that FTRTS-MOACO is the best in terms of DMR and fitness values and FTRTS-MAX and FTRTS-MIN, in order, are the best in terms of Makespan and LIMD. FTRTS-ACO performs well with all criteria but FTRTS-MOACO is still better.

7. CONCLUSION

Recently, improving the dependability of cloud environments for hosting real-time applications becomes a commercially and academically major concern. In this paper, a multi-objectives ACO based fault-tolerant real-time cloud scheduler, called FTRTS-MOACO, has been designed and implemented. The proposed scheduler aimed to optimize the deadline missing rate, Makespan, and load imbalance degree while preserving the principle of fault tolerance. A set of experiments are designed and performed to evaluate the performance of FTRTS-MOACO against a set of other fault-tolerant scheduling algorithms under different scenarios. The obtained results have shown the effectiveness and efficiency of the proposed scheduler regarding the different performance measures, particularly, DMR and Fitness. In the future, we intend to enhance the proposed scheduler to consider the highly restrictive requirements of hard real-time tasks in addition to improving both of Makespan and LIMD.

8. REFERENCES

[1] Gao, Y., Guan, H., Qi, Z., Hou, Y. and Liu, L., 2013. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, *79*(8), pp.1230-1242.

[2] Mohammed, B., Kiran, M., Maiyama, K.M., Kamala, M.M. and Awan, I.U., 2017. Failover strategy for fault tolerance in cloud computing environment. *Software: Practice and Experience*.

[3] Bilal, K., Khalid, O., Malik, S.U.R., Khan, M.U.S., Khan, S.U. and Zomaya, A.Y., 2016. Fault Tolerance in the Cloud. *Encyclopedia of Cloud Computing*, pp.291-300.

[4] Wang, J., Bao, W., Zhu, X., Yang, L.T. and Xiang, Y., 2015. FESTAL: fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds. *IEEE Transactions On Computers*, *64*(9), pp.2545-2558.

[5] Malik, S. and Huet, F., 2011, July. Adaptive fault tolerance in real time cloud computing. In *Services (SERVICES), 2011 IEEE World Congress on* (pp. 280-287). IEEE.

[6] Knight, J., 2012. *Fundamentals of Dependable Computing for Software Engineers*. CRC Press.

[7] Menychtas, A. and Konstanteli, K.G., 2012. Fault detection and recovery mechanisms and techniques for service oriented infrastructures. In *Achieving Real-Time in Distributed Computing: From Grids to Clouds* (pp. 259-274). IGI Global.

[8] Ganesh, A., Sandhya, M. and Shankar, S., 2014, February. A study on fault tolerance methods in cloud computing. In *Advance Computing Conference (IACC), 2014 IEEE International* (pp. 844-849). IEEE.

[9] Tawfeek, M.A., El-Sisi, A., Keshk, A.E. and Torkey, F.A., 2015, November. Cloud task scheduling based on ant colony optimization. *The International Arab Journal of Information Technology* 12 (2), pp.129-137.

[10] Chen, H. and Guo, W., 2015, June. Real-Time Task Scheduling Algorithm for Cloud Computing Based on Particle Swarm Optimization. In *International Conference on Cloud Computing and Big Data in Asia* (pp. 141-152). Springer, Cham.

[11] Selvaraj, C., Kumar, R.S. and Karnan, M., 2014. A survey on application of bio-inspired algorithms. International Journal of Computer Science and Information Technologies, 5(1), pp.366-70.

[12] Pandey, S., Wu, L., Guru, S.M. and Buyya, R., 2010, April. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced information networking and applications (AINA), 2010 24th IEEE international conference on* (pp. 400-407). IEEE.

[13] Xue, S., Li, M., Xu, X., Chen, J. and Xue, S., 2014. An ACO-LB algorithm for task scheduling in the cloud environment. *Journal of Software*, 9(2), pp.466-473.

[14] Bitam, S., 2012, February. Bees life algorithm for job scheduling in cloud computing. In *Proceedings of The Third International Conference on Communications and Information Technology* (pp. 186-191).

[15] Madni, S.H.H., Latiff, M.S.A., Abdullahi, M. and Usman, M.J., 2017. Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment. *PloS one*, 12(5), p.e0176321.

[16] Antony, S., Antony, S., Beegom, A.A. and Rajasree, M.S., 2012, September. Task scheduling algorithm with fault tolerance for cloud. In *Computing Sciences (ICCS), 2012 International Conference on* (pp. 180-182). IEEE.

[17] Latiff, M.S.A., Madni, S.H.H. and Abdullahi, M., 2016. Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm. *Neural Computing and Applications*, pp.1-15.

[18] Wang, X., Yeo, C.S., Buyya, R. and Su, J., 2011. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. *Future Generation Computer Systems*, *27*(8), pp.1124-1134.

[19] Zhao, L., Ren, Y., Xiang, Y. and Sakurai, K., 2010, September. Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems. In *High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on* (pp. 434-441). IEEE.

[20] Kessaci, Y., Melab, N. and Talbi, E.G., 2013. A Pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation. *Cluster Computing*, *16*(3), pp.451-468.

[21] Zhu, X., Yang, L.T., Chen, H., Wang, J., Yin, S. and Liu, X., 2014. Real-time tasks oriented energy-aware scheduling in virtualized clouds. *IEEE Transactions on Cloud Computing*, *2*(2), pp.168-180.

[22] Liu, S., Quan, G. and Ren, S., 2010, July. On-line scheduling of real-time services for cloud computing. In *Services (SERVICES-1), 2010 6th World Congress on* (pp. 459-464). IEEE.

[23] Jain, N., Menache, I., Naor, J.S. and Yaniv, J., 2015. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. *ACM Transactions on Parallel Computing*, *2*(1), p.3.

[24] Malik, S., Huet, F. and Caromel, D., 2012, December. Reliability aware scheduling in cloud computing. In *Internet Technology And Secured Transactions, 2012 International Conference for* (pp. 194-200). IEEE.

[25] Mahmood, A. and Khan, S.A., 2017. Hard Real-Time Task Scheduling in Cloud Computing Using an Adaptive Genetic Algorithm. *Computers*, 6(2), p.15.

[26] Corne, D.W., Reynolds, A. and Bonabeau, E., 2012. Swarm intelligence. In *Handbook of Natural Computing* (pp. 1599-1622). Springer Berlin Heidelberg.

[27] Dorigo, M. and Blum, C., 2005. Ant colony optimization theory: A survey. *Theoretical computer science*, *344*(2-3), pp.243-278.

[28] Merkle, D. and Middendorf, M., 2014. Swarm intelligence. In *Search methodologies* (pp. 213-242). Springer US.

[29] Dorigo, M. and Gambardella, L.M., 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, *1*(1), pp.53-66.

[30] Shyu, S.J., Lin, B.M. and Yin, P.Y., 2004. Application of ant colony optimization for no-wait flowshop scheduling problem to minimize the total completion time. *Computers & industrial engineering*, *47*(2), pp.181-193.

[31] Maniezzo, V. and Colorni, A., 1999. The ant system applied to the quadratic assignment problem. *IEEE Transactions on knowledge and data engineering*, *11*(5), pp.769-778.

[32] Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D. and Freund, R.F., 1999. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of parallel and distributed computing*, *59*(2), pp.107-131.