International Journal of Communication Networks and Information Security (IJCNIS) Vol. 11, No. 2, August 2019

Load Balancing Using Dynamic Ant Colony System Based Fault Tolerance in Grid Computing

Saufi Bukhari[1], Ku Ruhana Ku-Mahamud[1] and Hiroaki Morino[2]

[1]School of Computing, UUM College of Arts and Sciences, Universiti Utara Malaysia, Malaysia
[2]Graduate School of Engineering and Science, Shibaura Institute of Technology, Japan

Abstract: Load balancing is often disregarded when implementing fault tolerance capability in grid computing. Effective load balancing ensures that a fair amount of load is assigned to each resource, based on its fitness rather than assigning a majority of tasks to the most fitting resources. Proper load balancing in a fault tolerance system would also reduce the bottleneck at the most fit resources and increase utilization of other resources. This paper presents a fault tolerance algorithm based on ant colony system, that considers load balancing based on resource fitness with resubmission and checkpoint technique, to improve fault tolerance capability in grid computing. Experimental results indicated that the proposed fault tolerance algorithm has better execution time, throughput, makespan, latency, load balancing and success rate.

Keywords: Load Balancing, Task Scheduling, Task Checkpoint, Task Resubmission, Ant Colony System

1. Introduction

Grid computing has been in the industry for many years providing intensive parallel and distributed computing capabilities to process large tasks. It is also one of the sub-capabilities in recent distributed systems such as cloud and cluster computing. Generally, when a system consists of multiple independent computing resources based in different locations, it is impossible to prevent failures from happening. The only solutions are to mitigate the failure when it happens or prevent it from happening. There are many types of failures as described in [1] such as network failure (e.g.: packet loss and corruption), physical failure (e.g.: damaged CPU and storage drive), user termination, service and protocol failure. Any disruption in the processing machine would definitely lead to delay in response time for the users due to submitted tasks cannot be processed according to expected completion time and resources cannot be released to process subsequent tasks in the queue [2]. As a result, stagnation may occur and the throughput will be greatly degraded due to limited resources available to process the tasks in the queue. In addition to that, resource utilization will be reduced because initial scheduling and resource assignment is affected by the occurrence of failure.
To minimize this problem, effective fault tolerance should be implemented to identify occurrence of failures more accurately during runtime, ensure reliable execution and preserve the great potential of computational grids [3, 4]. Fault tolerance is defined as a Nondeterministic Polynomial (NP)-complete problem [5, 6] which means that there are more than one suboptimal solutions to solve problems in a polynomial time [7, 8]. Typically, approximate (heuristic) algorithms such as Genetic Algorithm (GA) [9], Simulated Annealing (SA) [10], Tabu Search (TS) [11] and recently Ant Colony Optimization (ACO) [12] are used to solve these problems. These algorithms are used to construct the best solution by moving from one solution to another dynamically. A feasible and optimal solution could be produced at a time, but it will not be the optimal solution at another time due to dynamically changing environment.
ACO is an example of biologically-inspired algorithm that provides an adaptive concept for solving optimization problems and designing metaheuristics algorithms [13, 14]. The concept is almost similar to other heuristics algorithms whereby the best solution is constructed by a group of ants within the colony. Each ant is responsible to construct individual solution and all the individual solutions will be consolidated to build the best or optimal solution. The solution is represented by the pheromone intensity where the following ants will refer to its strength to choose the optimal path. ACO is very effective when it comes to enhancing scheduling and load balancing in grid computing, but the optimal path finding capability can be further upgraded to allow new path to alternative resource to be constructed in the presence of a failure, as illustrated in Figure 1(a-d) respectively.

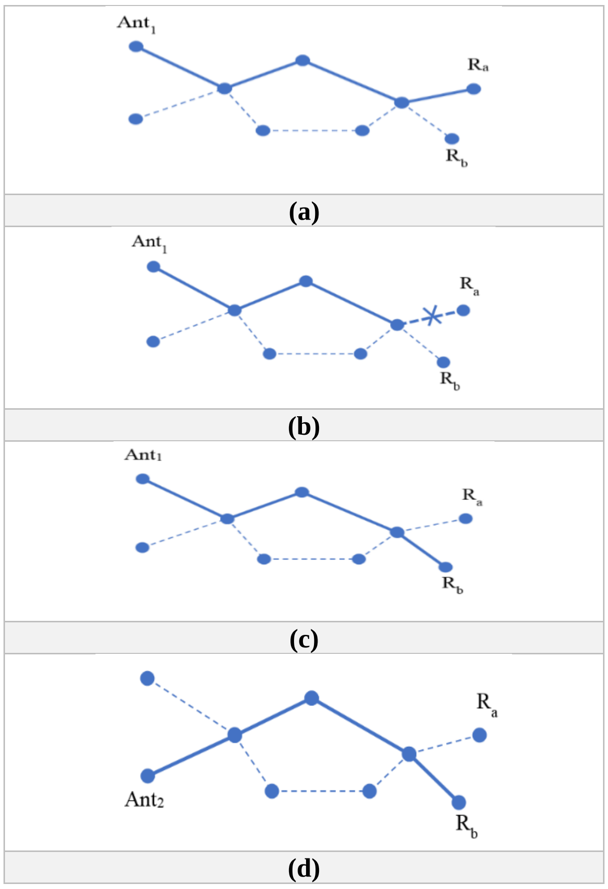


Figure 1(a-d). Illustration of the way ants find an alternative resource during failure

Figure 1(a) shows that Ant_1 constructs a path to resource R_a and assigns the task. Then, in Figure 1(b), the Resource R_a fails to complete task execution. In Figure 1(c), Ant_1 constructs another path using previous path to alternative resource R_b to assign failed task. Finally, as in Figure 1(d), the next Ant_2 uses previously constructed path to resource R_b to assign the next task.

This paper proposes Enhanced Dynamic ACO-based Fault Tolerance (EDAFT) in grid computing extended from the Ant Colony System (ACS) scheduling algorithm, that focuses on load balancing based on resource fitness in addition to providing fault tolerance. Section 2 covers some of the related works and followed by details on the proposed algorithm in the Section 3. All the experimental results are analyzed and presented in the Section 4, whereas the last section concludes the paper.

2. Related Works

Load balancing using enhanced ACO was proposed by [15] to effectively balance task allocation to all available resources in a grid computing system. In the beginning, the initial pheromone will be calculated for each combination of tasks and resources by considering processing capability, size of task, bandwidth and current load. Once calculated, the initial pheromone value will be stored in a two-dimensional matrix and referred to by an ant to select the best resource. For each task in queue, an ant will be spawned to search for the resource with the highest pheromone in the matrix. When the resource is identified, a task will be assigned, and a global pheromone update will be performed to all entries associated with the current resource. Once the execution is completed, the resource will be released to process another task. The results showed that the proposed algorithm is able to increase resource utilization and load balancing. Notwithstanding the results, it is possible to adapt this concept in a fault tolerance algorithm to improve resource utilization and load balancing in a faulty grid computing environment.

Trust-based ACO for grid resource scheduling was proposed by [16] with the goals of balancing the workload and reducing task completion time. They proposed a mechanism called ‘trust’ to reward the success or penalize the failure and being applied during the global pheromone update process. Reward is a positive constant value which will increase the amount of calculated pheromone while penalty is a negative constant value that will decrease the amount of calculated pheromone. This mechanism is claimed to reduce the possibility of failure during task execution as the task assignment will also consider the amount of pheromone at each available resource. In terms of failure handling mechanism, each failed task will be reinserted into the queue for a reallocation process in addition to applying the penalty. In the end, the algorithm ensures that all submitted tasks complete the execution. Despite the effectiveness of applying the trust mechanism and reallocation process, due to the lack of checkpoint technique that requires all the failed tasks to be reprocessed from the beginning, it may lead to higher latency and makespan.

[17] proposed an algorithm that combines ACO with Genetic Algorithm (GA) to reduce performance degradation due to the uncontrolled nature of the metaheuristics of an ACO. The GA is used to determine whether to increase or decrease the pheromone update parameters. Before forming subsets, ants will perform resource selection randomly using ACO. Then, each subset will be evaluated using the GA algorithm, to spot the lowest estimated error and will be sorted ascendingly. Tasks will be assigned to the subset with the lowest estimated error and followed by the application of the pheromone update once the execution is completed. Resources within the subset with the lowest estimated error will have a higher probability of getting tasks assigned. It is claimed that the proposed algorithm is suitable to be applied during task scheduling. Notwithstanding the performance of the proposed algorithm in reducing the possibility of failure, it could be further enhanced by incorporating effective recovery techniques during occurrence of failure.

Fault tolerance ACO using checkpoints was proposed by [18] to solve fault and load balancing problems by finding the optimal resource as well as detecting the occurrence of failure during task execution. A component called fault index manager is applied to record the failure history that is used as a reference in the next task assignment. When failure occurs, failed tasks will be rescheduled to alternative optimal resources using checkpoint technique from the last saved state instead of from the beginning. In terms of load balancing aspect, tasks will have higher possibility to be assigned to the resources with a low workload. The workload is indicated by the pheromone value of each resource which will continuously be updated during the pheromone update process. Although the proposed algorithm looks promising, it is just a conceptual algorithm which has not been developed and validated to proof its claimed performance.

Task scheduling with fault tolerance in grid computing using ACO was proposed by [19] that combines checkpoint and resubmission techniques. In the fault tolerance architecture, they proposed a fault index that is maintained by the fault index handler. The checkpoint handler works closely with a fault index handler to determine the resource failure rate to control the checkpoint interval and the number of checkpoints which is claimed to minimize task processing time and increase throughput. The checkpoint handler interacts with a scheduler to perform unconditional task scheduling that includes both initial submission and resubmission after failure. The results showed that the proposed algorithm reduces makespan, increases throughput and the average turnaround time. Despite having good performance, the consideration of resource load alone is believed to be not an effective method to determine the resource fitness and may lead to higher chance of execution failure.

Dynamic ACS-based fault tolerance in grid computing was proposed by [20] to reduce the execution time and task processing time, and to increase execution success rate. The proposed algorithm consists of a resubmission to alternative resources using checkpoint technique and consideration of resource execution history during the pheromone update process, to ensure that all failed tasks can complete execution. For every checkpoint, the execution status will be recorded and used during the pheromone update process to penalize unfit resources so that they become less attractive to allow the subsequent ants to explore other fit resources. In addition to that, once the execution is completed, another round of pheromone update will be applied to further reduce the pheromone level. The experimental results showed that the proposed algorithm gives a better average execution time per task and execution success rate. It is also possible to control the penalty by introducing a trust factor, so that resources that complete the execution will be more attractive without disregarding the load balancing aspect.

ACO has been one of the most adaptable algorithms for dynamic scheduling in distributed systems. Integration with other techniques such as task resubmission and checkpoint is essential to further enhance its fault tolerance capability. In addition to that, execution history and trust factor can also be considered to provide better scheduling decisions to reduce the turnaround time and failure rate without significant impact to the performance and load balancing of the system.

3. Proposed Algorithm

Enhanced Dynamic ACS-based Fault Tolerance (EDAFT) is the extended version of algorithm proposed by [20] that is inspired by the foraging behavior of an ant colony to search for the food source by constructing optimal path between the nest and food source. This analogy is similar to the process of constructing optimal path between tasks and resources in grid computing system. In the proposed algorithm, this process is further extended for ants to have the ability to perform resource researching during the checkpoint-based resubmission process to assign any failed task to alternative resources with higher probability of success. To further improve the pheromone update technique, a trust factor is introduced to reward fit resources, or penalize unfit resources, with a consideration of the execution history to control the pheromone reduction or increase. The improved pheromone update formula is expected to properly control the task assignment based on resource fitness which could eventually reduce the possibility of failure.

During the initial task submission, each resource should have pre-defined parameters such as processor speed, current load, and bandwidth and number of processing elements. All these parameters will be used to calculate the initial pheromone value (PV_{rj}) for each combination of resource r and task j. The initial pheromone value formula is given by the following equation (1):

$$PV_{rj} = \left[\frac{S_j}{bandwidth_r} + \frac{C_j}{MIPS_r(1-load_r)}\right]^{-1} \quad (1)$$

Where Sj is the size and Cj is the required computation power of a given task j, $bandwidth_j$ is the available bandwidth of resource r, $MIPS_r$ is the processor speed, and $load_r$ is the current load at resource r. Note that the initial pheromone value is assigned during initialization, but after that, it is considered as a resource pheromone value. Since the initial pheromone value is calculated for each combination of resource and task, this information is stored in a PV_{matrix} as in (2):

$$PV_{matrix} = \begin{bmatrix} PV_{1,1} & PV_{1,2} & PV_{1,n-1} & PV_{1,n} \\ PV_{2,1} & PV_{2,2} & PV_{2,n-1} & PV_{2,n} \\ PV_{m-1,1} & PV_{m-1,2} & PV_{m-1,n-1} & PV_{m-1,n} \\ PV_{m,1} & PV_{m,2} & PV_{m,n-1} & PV_{m,n} \end{bmatrix} \quad (2)$$

Where n is total number of tasks and m is total number of resources. PV_{matrix} is a logical form of ant topology whereby an ant would move from one index to another index to find the best resource for task assignment. It is assumed that all the resources are interconnected which means that if the task originates from a specific resource, it can be assigned to all other available resources. Each row in PV_{matrix} represents the list of possible tasks for resource r while each column represents the list of possible resources for task j. The largest pheromone value in each column will be considered by the ants as the most fit resource and the task will be forwarded to the resource with highest pheromone for processing.

As soon as the task is assigned, the pheromone value in the PV_{matrix} will be updated by the global pheromone update (3) to reduce the amount of pheromones assigned to the current resource, so that it becomes less attractive by the next ant and leads to the exploration of other resources. τ_{rj} is the amount of pheromones on the resource, while $\Delta\tau_{rj}$ is $1/L_{best}$, where L_{best} denotes the length of global best tour or otherwise (no global best tour found), $\Delta\tau_{rj}$=0.

$$\tau_{rj} = (1-\rho)\cdot\tau_{rj} + \rho\cdot\Delta\tau_{rj} \quad (3)$$

ρ is the evaporation rate that is dynamically controlled by using the following formula (4) with m and n as the total number of resources and tasks respectively:

$$\rho = \left[\left(\frac{n}{m}\right)^2\right]^{-1} \quad (4)$$

A typical ACS algorithm consists of global and local pheromone updates. In EDAFT, the local pheromone update is improved to include a trust factor so that more pheromone is added should the resource complete task execution or otherwise the existing pheromone evaporates. The improved global pheromone update (5) is given as follows:

$$\tau_{rj} = (1-\rho)\cdot\tau_{rj} + [\rho\cdot\tau_0(R_H)]^C \quad (5)$$

Where ρ is the evaporation rate, τ_{rj} is the current pheromone intensity for resource r, τ_0 is the initial pheromone value of resource r, C is the trust factor defined by either task completion (C = 1.5) or task failure (C = 1.0). The defined value of C in this experiment provides the lowest load balancing standard deviation using equation (11). The trust factor may vary depending on the level of trust sensitivity to be applied, in which too much penalty may cause unfit resources to never get assigned with tasks after failure or too much incentive may cause fit resources to be assigned with most of the tasks. R_H is the average weighted execution history of resource r and is calculated by equation (6):

$$R_H(i) = \begin{cases} R_T(i) = \frac{CP_{success}}{CP_{failed}+CP_{success}}, i = 0 \\ (1-\alpha)\cdot R_T(i) + \alpha\cdot R_H(i-1), i > 0 \end{cases} \quad (6)$$

Where $R_T(i)$ is the current execution history at take i, $CP_{success}$ indicates the current successful checkpoint call, and CP_{failed} is the current failed checkpoint, at resource r respectively. For each resource r, i is initially set to 0 and will be incremented by 1 for each local pheromone update process, $R_H(i\text{-}1)$ is the previously recorded execution history and α is the degree of weighting decrease set to 0.5. The execution history (also known as resource fitness) will be used to control the quantity of pheromones to be evaporated, or strengthened, at a respective resource and eventually helps the following ants to identify the best resources during task assignment; the better the execution history, the higher the number of tasks assigned. Figure 2 illustrates the high-level workflow of EDAFT as proposed by [20, 21] with the improved process bolded for clarity. An ant will be generated for each task in the queue to perform resource searching based on pheromone values. Before the first task in queue can be submitted, the initial pheromone value will be calculated to determine the state of all resources. The resource selection will be performed based on the pheromone levels, either from the initial pheromone calculation or the pheromone update process. Once the task is assigned to any resource, the ant will apply global pheromone update to reduce the amount of pheromone so that the resource becomes less attractive for the next ant. Each assigned task will be divided into several time-based checkpoints recorded during execution. In the event of failure, the task will undergo rescheduling process and will be assigned to alternative resource from the last saved checkpoint and a local pheromone update with penalty will be applied to the resource that failed to reduce the pheromone intensity. If the execution is successful, the local pheromone update with incentive will be applied to the resource to increase the pheromone. In both execution failure and execution success scenarios, the resource will be released for the next task assignment after the pheromone update process.

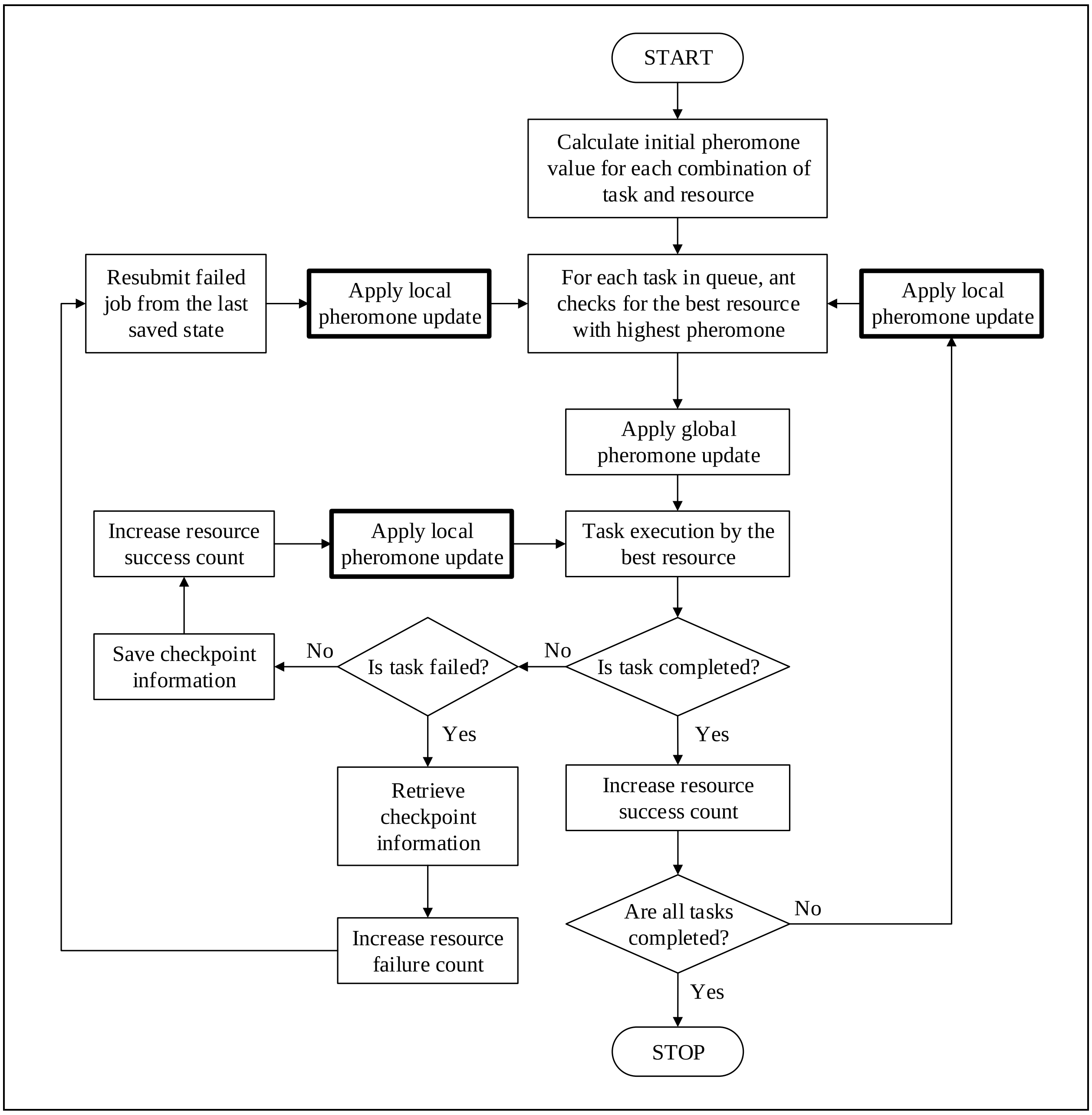

START
Calculate initial pheromone value for each combination of task and resource
Resubmit failed job from the last saved state
Apply local pheromone update
For each task in queue, ant checks for the best resource with highest pheromone
Apply local pheromone update
Apply global pheromone update
Increase resource success count
Apply local pheromone update
Task execution by the best resource
Save checkpoint information
No
Is task failed?
No
Is task completed?
Yes
Yes
Retrieve checkpoint information
Increase resource success count
Are all tasks completed?
No
Increase resource failure count
Yes
STOP

Figure 2. High-level workflow of EDAFT

4. Experimental Design and Results

There are various performance metrics used to measure the proposed algorithm, which include execution time, throughput, makespan, latency, load balancing and success rate. Execution time (equation 7) is measured from the moment the first task is submitted to the system, $SubmissionTime_1$ to undergo scheduling and execution process, until the last task n is completely processed, $CompletionTime_n$.

$$ExecutionTime = CompletionTime_n - SubmissionTime_1 \quad (7)$$

Throughput (equation 8) is calculated by dividing the total number of tasks, n, with the total time taken to completely process all tasks [19].

$$Throughput = \frac{n}{ExecutionTime} \quad (8)$$

The average turnaround time per task is also considered as the average execution time per individual task, from the moment the task received by the resource until completely processed. As shown in equation (9), it is measured by summing the execution time for each individual task and dividing the result by the total number of tasks n.

$$AverageTurnaroundTime = \frac{\sum_{j=1}^{n}(CompletionTime_j - ArrivalTime_j)}{n} \quad (9)$$

Average latency per task measures the waiting time for each task to be processed by the assigned resource, from the moment the task is submitted to the queue until arrival at the resource. As depicted in equation (10), total latency for all tasks is divided by the total number of tasks n.

$$AverageLatency = \frac{\sum_{j=1}^{n}(ArrivalTime_j - SubmissionTime_j)}{n} \quad (10)$$

Load balancing is measured by calculating the standard deviation of the initially assigned fitness rate and actual ratio of total processed tasks, as in equation (11). Population standard deviation formula is used as a base formula, as follows:

$$\sigma_r = \sqrt{\frac{\sum(X_r - \mu_r)^2}{N}} \quad (11)$$

Where X_r is the percentage of total tasks executed by resource r, μ_r is the fitness rate of resource r and N is the total number of resources. Instead of using the mean, the percentage of total tasks processed by a resource is used to measure the effectiveness of the load balancing aspect. The lower the standard deviation is, the better load balancing the resource has. For a more accurate measurement, the processing capability of each resource should be identical, while the task and output size should be within an acceptable range. The proposed formula is suitable to measure the load balancing at the end of simulation but is not intended to measure the load balancing during runtime. Finally, the execution success rate (equation 12) calculates the total number of successful checkpoints, $CP_{success}$ over the total number of recorded checkpoints (CP_{failed} + $CP_{success}$).

$$SuccessRate = \frac{\sum CP_{success}}{\sum(CP_{failed} + CP_{success})} \quad (12)$$

To validate the performance of the proposed EDAFT algorithm in the presence of failure, pseudorandom algorithm is used to randomly assign resource fitness within a defined range. In this case, the range of resource fitness is defined as having a value between 50% and 100%, as used by [2]. It is important to define the range of resource fitness to validate the effectiveness of fault tolerance algorithm in handling different possibilities of failures. On the other hand, other resources and task parameters are adopted from [19] as shown in Table 1.

Table 1. Simulation parameters

Parameter	Value
No. of resources	100
No. of tasks	5000
PE rating	50 MIPS
Bandwidth	5000 B/S
Machine per resource	1
PE per machine	2
Task length	50000
File size	100 + (10-40%)
Output size	250 + (10-50%)

The proposed algorithm was compared with TACO [16], FTACO [18], ACOwFT and ACO [19] where all algorithms were redeveloped based on published pseudocodes, formulations and flowcharts. They are suitable to be used for validation, because the EDAFT algorithm is inspired by all algorithms in terms of the fault tolerance techniques. All the experiments are simulated in a JAVA based simulator, known as GridSim Toolkit, because it provides a comprehensive simulated grid environment within which most of the components are included. Each algorithm is executed 10 times for each fault range and the average is taken for a more precise measurement. The list of performance metrics includes execution time, throughput, average makespan and latency per task, load balancing and execution success rate.

The execution time for all algorithms are almost similar when there is no failure (refer Figure 3). However, as the failure rate increases, the execution time for EDAFT is the lowest among all and followed by ACOwFT and FTACO respectively. The results also suggest that ACO and TACO have significantly longer execution time because the application of checkpoint technique provides significant reduction of execution time as the failed task does not need to be re-executed from the beginning. This technique is also effective especially when the size of each task is big and the expected time to completely execute each task is long.

The results of throughput for all algorithms is depicted in Figure 4. Throughput is mainly influenced by the execution time and total amount of completed task. In this case, the throughput for ACO and TACO are in agreement with the execution time in Figure 3. At 0% failure rate, the throughput for all algorithms is at the highest, and gradually decreased as the failure rate increased. The results also suggest that EDAFT has the lowest reduction of throughput from 10% to 50% failure rate as compared to the other algorithms. It is crucial to preserve the execution time despite in the presence of failure to ensure that the throughput can also be preserved.

As presented in Figure 5, the average execution time for EDAFT, FTACO and ACOwFT are almost similar as compared to TACO and ACO. This is where the checkpoint technique plays a role as it allows each failed task to be executed from the last saved state, instead of from the beginning, which eventually reduces the turnaround time per individual task. The results also suggest that EDAFT, FTACO and ACOwFT are able to control the turnaround time using checkpoint technique in the presence of failures so that each individual task can be executed completely in a timely manner.

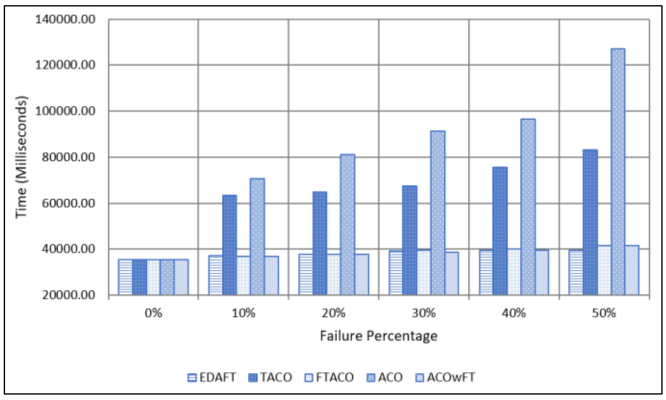
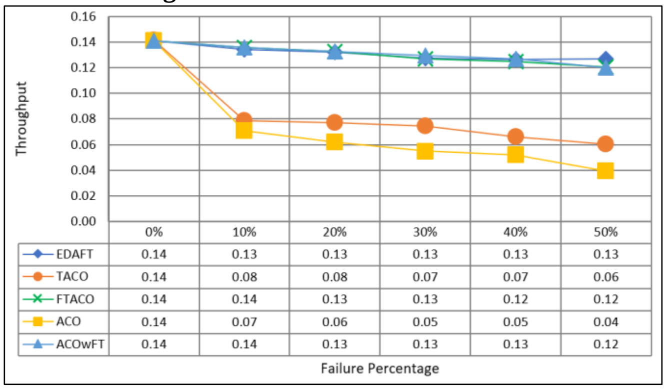
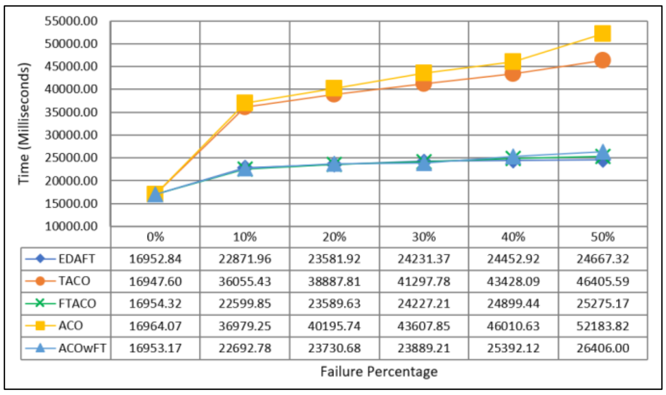


Figure 3. Results of execution time



	0%	10%	20%	30%	40%	50%
EDAFT	0.14	0.13	0.13	0.13	0.13	0.13
TACO	0.14	0.08	0.08	0.07	0.07	0.06
FTACO	0.14	0.14	0.13	0.13	0.12	0.12
ACO	0.14	0.07	0.06	0.05	0.05	0.04
ACOwFT	0.14	0.14	0.13	0.13	0.13	0.12

Figure 4. Results of throughput

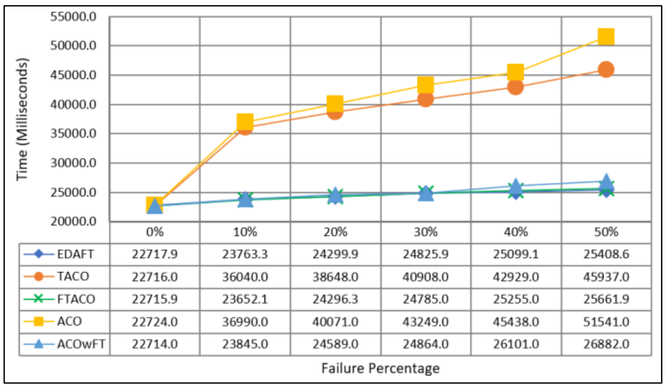


	0%	10%	20%	30%	40%	50%
EDAFT	16952.84	22871.96	23581.92	24231.37	24452.92	24667.32
TACO	16947.60	36055.43	38887.81	41297.78	43428.09	46405.59
FTACO	16954.32	22599.85	23589.63	24227.21	24899.44	25275.17
ACO	16964.07	36979.25	40195.74	43607.85	46010.63	52183.82
ACOwFT	16953.17	22692.78	23730.68	23889.21	25392.12	26406.00

Figure 5. Results of average turnaround time per task

The average latency as shown in Figure 6 is also having the same pattern as average turnaround time in Figure 5. The results suggest that ACO has the highest latency and followed by TACO in which both algorithms do not apply checkpoint technique. The latency can also be controlled by properly distributing the tasks to all available resources to avoid bottleneck where queues for some resources are longer than the average queue length.

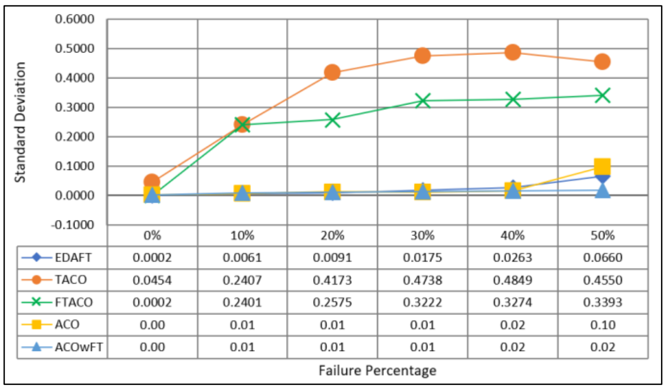
Load balancing is essential to measure how well the task distribution is performed. As shown in Figure 7, ACOwFT and ACO have relatively the lowest standard deviation because the task assignment is done based on the current load of each resource. As for EDAFT with almost the same performance as ACOwFT, the task assignment is performed by considering the execution history in determining the fitness of the resource and balancing the load. The closer the standard deviation to 0, the better the load balancing is. In other words, without even knowing the fitness of a specific resource in the first place, the proposed algorithm is able to apply heuristic capability to determine the fitness based on execution history while preserving the resource utilization.



	0%	10%	20%	30%	40%	50%
EDAFT	22717.9	23763.3	24299.9	24825.9	25099.1	25408.6
TACO	22716.0	36040.0	38648.0	40908.0	42929.0	45937.0
FTACO	22715.9	23652.1	24296.3	24785.0	25255.0	25661.9
ACO	22724.0	36990.0	40071.0	43249.0	45438.0	51541.0
ACOwFT	22714.0	23845.0	24589.0	24864.0	26101.0	26882.0

Figure 6. Results of average latency per task

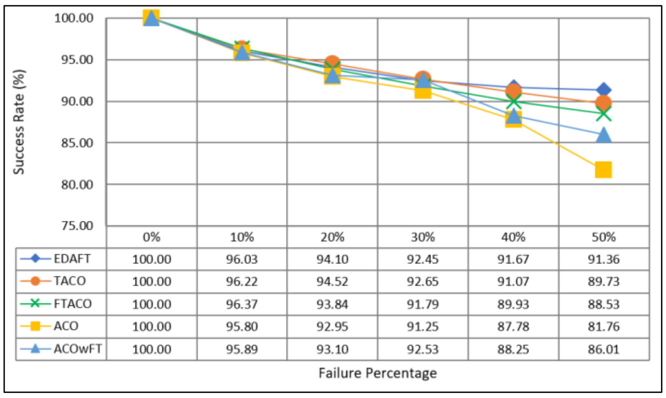
In addition to that, the trust factor is useful to reward fit resources or penalize unfit resources based on task execution status. Coupling both components provides a more effective pheromone update process that can tackle load balancing and the execution success rate.



	0%	10%	20%	30%	40%	50%
EDAFT	0.0002	0.0061	0.0091	0.0175	0.0263	0.0660
TACO	0.0454	0.2407	0.4173	0.4738	0.4849	0.4550
FTACO	0.0002	0.2401	0.2575	0.3222	0.3274	0.3393
ACO	0.00	0.01	0.01	0.01	0.02	0.10
ACOwFT	0.00	0.01	0.01	0.01	0.02	0.02

Figure 7. Results of load balancing

In any fault tolerance system, the ultimate aim is to maintain the execution success rate without disregarding the performance. Figure 8 shows that EDAFT has a higher success rate compared to the other algorithms. Surprisingly, TACO has the second highest success rate because it assigns most of the tasks to fit resources rather than unfit resources. It is obvious that whenever most of the tasks are assigned to the most fit resources, it would increase the possibility of success.



	0%	10%	20%	30%	40%	50%
EDAFT	100.00	96.03	94.10	92.45	91.67	91.36
TACO	100.00	96.22	94.52	92.65	91.07	89.73
FTACO	100.00	96.37	93.84	91.79	89.93	88.53
ACO	100.00	95.80	92.95	91.25	87.78	81.76
ACOwFT	100.00	95.89	93.10	92.53	88.25	86.01

Figure 8. Results of success rate

The drawback of TACO is that the latency and makespan will be significantly higher because fit resources will have a longer queue. In addition to that, ACOwFT and ACO have the lowest success rate because by referring to the load alone is not sufficient to determine the fitness of the resource. Some resources may have low load because they are not actively executing tasks. However, for EDAFT, it can maintain the success rate in addition to maintaining throughput and providing better load balancing, lower latency and makespan. It is also discovered that by reducing the trust value *C* in the presence of failure, EDAFT can achieve a higher success rate, but will also increase the load balancing standard deviation. Thus, it is important to define the most optimal trust value to achieve either the best execution success rate or the best load balancing.

5. Conclusion

It can be concluded that EDAFT has the best overall performance compared to the other algorithms in terms of execution time, throughput, average turnaround time, average latency and execution success rate. However, in terms of load balancing, ACOwFT has the best performance with a slight difference compared to ACO and EDAFT. Despite EDAFT having an overall good performance, it can be further enhanced by including a temporary suspension so that a recently failed resource will not be assigned a task until it is recovered from failure. This capability may be effective, especially when the size of an individual task is large at a point of time, when most resources are busy processing a large task and only the current suspended task is idle. Crucially, the decision is important to not assign the task immediately to the recently failed resource to reduce the possibility of another failure and to select the resource that can complete current execution in the least time and has a good fitness. Additionally, the length of suspension is also an important aspect so that a sufficient length of suspension can be applied to preserve the load balancing and execution success rate.

6. Acknowledgements

The authors thank the Ministry of Higher Education Malaysia in funding this study under the Trans Disciplinary Research Grant Scheme (TRGS), S/O code 13164 and Research and Innovation Management Centre (RIMC), Universiti Utara Malaysia, Kedah for the administration of this study.

References

[1] D. Rakheja, P. Kaur, A. Rkheja, “Performance evaluation of resource scheduling and fault tolerance in grid,” International Journal of Computer and Communication System Engineering, Vol. 1, No. 01, pp. 15-19, 2014.

[2] M. Amoon, “A fault tolerance scheduling system based on checkpointing for computational grids,” International Journal of Advanced Science and Technology, Vol. 48, pp. 115-124, 2012.

[3] M. Nandagopal, V. R. Uthariaraj, “Fault tolerant scheduling strategy for computational grid environment,” International Journal of Engineering Science and Technology, Vol. 2, No. 9, pp. 4361-4372, 2010.

[4] P. Keerthika, N. Kasthuri, “An efficient fault tolerant scheduling approach for computational grid,” American Journal of Applied Sciences, Vol. 9, No. 12, pp. 2046-2051, 2012.

[5] C. Glaßer, A. Pavan, S. Travers, “The fault tolerance of NP-hard problems,” Language and Automata Theory and Applications, Springer Berlin Heidelberg, 2009.

[6] H. J. A. Nasir, K. R. Ku-Mahamud, E. Kamioka “Enhanced ant-based routing for improving performance of wireless sensor network,” International Journal of Communication Networks and Information Security, Vol. 9, No. 3, pp. 386-392, 2017.

[7] C. Blum, A. Roli, “Metaheuristics in combinatorial optimization: Overview and conceptual comparison,” Journal of ACM Computing Surveys, Vol. 35, No. 3, pp. 268-308, 2003.

[8] Z. Pooranian, M. Shojafar, J. H. Abawajy, M. Singhal, “GLOA: A new job scheduling algorithm for grid computing,” International Journal of Artificial Intelligence and Interactive Multimedia, Vol. 2, No. 1, pp. 59-64, 2013.

[9] F. Werner, “Genetic algorithms for shop scheduling problems: A survey,” Preprint, Vol. 11, No. 31, pp. 1-66, 2011.

[10] S. W. Lin, F. Y. Vincent, “A simulated annealing heuristic for the team orienteering problem with time windows,” European Journal of Operational Research, Vol. 217, No. 1, pp. 94-107, 2012.

[11] F. Glover, M. Laguna, “Tabu Search*,” Handbook of Combinatorial Optimization, Springer New York, 2013.

[12] K. R. Ku-Mahamud, M. M. Alobaedy, “New heuristic function in ant colony system for job scheduling in grid computing,” Mathematical Methods for Information Science and Economics, WSEAS, Montreux, 2012.

[13] M. Dorigo, T. Stützle, “Ant colony optimization,” MIT Press, Cambridge, 2004.

[14] M. H. Ferdaus, M. Murshed, R. N. Calheiros, R. Buyya, “Virtual machine consolidation in cloud data centers using ACO metaheuristic,” Euro-Par 2014 Parallel Processing, Springer International Publishing, Porto, 2014.

[15] H. J. A. Nasir, K. R. Ku-Mahamud, “Grid load balancing using ant colony optimization,” Second International Conference on Computer and Network Technology, Bangkok, Thailand, pp. 207-211, 2010.

[16] H. Wenming, D. Zhenrong, W. Peizhi, “Trust-based ant colony optimization for grid resource scheduling,” Third International Conference on Genetic and Evolutionary Computing, Guilin, China, pp. 288-292, 2009.

[17] S. Mandloi, H. Gupta, “Adaptive job scheduling for computational grid based on ant colony optimization with genetic parameter selection,” International Journal of Advanced Computer Research, Vol. 3, No. 9, pp. 66-71, 2013.

[18] T. Prashar, Nancy, D. Kumar, “Fault tolerant ACO using checkpoint in grid computing,” International Journal of Computer Applications, Vol. 98, No. 10, pp. 44-49, 2014.

[19] H. Idris, A. E. Ezugwu, S. B. Junaidu, A. O. Adewumi, “An improved ant colony optimization algorithm with fault tolerance for job scheduling in grid computing systems,” PLOS ONE, Vol. 12, No. 5, pp. 1-24, 2017.

[20] S. Bukhari, K. R. Ku-Mahamud, H. Morino, “Fault tolerance grid scheduling with checkpoint based on ant colony system,” Journal of Computer Science, Vol. 13, No. 8, pp. 363-370, 2017.

[21] S. Bukhari, K. R. Ku-Mahamud, H. Morino, “Dynamic ACO-based fault tolerance in grid computing,” International Journal of Grid and Distributed Computing, Vol. 10, No. 12, pp. 117-124, 2017.