AN ENHANCED ANT COLONY SYSTEM ALGORITHM FOR DYNAMIC FAULT TOLERANCE IN GRID COMPUTING

SAUFI BIN BUKHARI

DOCTOR OF PHILOSOPHY
UNIVERSITI UTARA MALAYSIA
2020

Awang Had Salleh Graduate School of Arts And Sciences

Universiti Utara Malaysia

PERAKUAN KERJA TESIS / DISERTASI
(Certification of thesis / dissertation)

Kami, yang bertandatangan, memperakukan bahawa
(We, the undersigned, certify that)

SAUFI BUKHARI

calon untuk Ijazah **PhD**
(candidate for the degree of)

telah mengemukakan tesis / disertasi yang bertajuk:
(has presented his/her thesis / dissertation of the following title):

"AN ENHANCED ANT COLONY SYSTEM ALGORITHM FOR DYNAMIC FAULT TOLERANCE IN GRID COMPUTING"

seperti yang tercatat di muka surat tajuk dan kulit tesis / disertasi.
(as it appears on the title page and front cover of the thesis / dissertation).

Bahawa tesis/disertasi tersebut boleh diterima dari segi bentuk serta kandungan dan meliputi bidang ilmu dengan memuaskan, sebagaimana yang ditunjukkan oleh calon dalam ujian lisan yang diadakan pada : ***30 April 2020.***
That the said thesis/dissertation is acceptable in form and content and displays a satisfactory knowledge of the field of study as demonstrated by the candidate through an oral examination held on:
April 30, 2020.

Pengerusi Viva: *(Chairman for VIVA)*	**Prof. Dr. Wan Rozaini Sheik Osman**	Tandatangan *(Signature)* Rozaini
Pemeriksa Luar: *(External Examiner)*	**Prof. Dr. Ali Selamat**	Tandatangan *(Signature)* [illegible]
Pemeriksa Dalam: *(Internal Examiner)*	**Dr. Mohamad Farhan Mohamad Mohsin**	Tandatangan *(Signature)* [illegible]
Nama Penyelia/Penyelia-penyelia: *(Name of Supervisor/Supervisors)*	**Prof. Dr. Ku Ruhana Ku Mahamud**	Tandatangan *(Signature)* [illegible]
Nama Penyelia/Penyelia-penyelia: *(Name of Supervisor/Supervisors)*	**Prof. Dr. Hiroaki Morino**	Tandatangan *(Signature)* Hiroaki Morino

Tarikh:
(Date) ***April 30, 2020***

Permission to Use

In presenting this thesis in fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the University Library may make it freely available for inspection. I further agree that permission for the copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in whole or in part, should be addressed to:

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUM College of Arts and Sciences
Universiti Utara Malaysia
06010 UUM Sintok

Abstrak

Toleransi sesar di dalam pengkomputeran grid membolehkan sistem terus beroperasi walaupun kegagalan berlaku. Kebanyakan algoritma toleransi sesar memfokus kepada teknik pengendalian kegagalan seperti pemprosesan semua kerja, semakan titik, pereplikaan kerja, penalti, dan pemindahan kerja. Sistem koloni semut (ACS), salah satu variasi pengoptimuman koloni semut (ACO), adalah salah satu algoritma yang baik untuk toleransi sesar disebabkan kebolehannya menyesuaikan diri dengan masalah pengoptimuman kombinatorik statik dan dinamik. Walau bagaimanapun, algoritma ACS tidak mengambil kira kecergasan sumber ketika menjadualkan kerja sekaligus menyebabkan pengimbanan beban yang tidak cekap dan kadar kejayaan pelaksanaan yang rendah. Kajian ini mencadangkan toleransi sesar ACS secara dinamik dengan penggantungan (DAFTS) di dalam pengkomputeran grid yang memfokus kepada penyediaan teknik toleransi sesar yang efektif untuk menambahbaik kadar kejayaan pelaksanaan dan pengimbangan beban. Algoritma yang telah dicadangkan terdiri daripada kadar evaporasi secara dinamik, proses penjadualan berdasarkan kecergasan sumber, pengemaskinian feromon yang dipertingkat dengan faktor kepercayaan dan penggantungan, dan pemprosesan semula menggunakan semakan titik. Rangka kerja kajian merangkumi empat fasa iaitu mengenalpasti teknik toleransi sesar yang akan digunakan, meningkatkan proses penugasan sumber dan penjadualan kerja, menambahbaik algoritma toleransi sesar dan menilai kecekapan algoritma yang dicadangkan. Algoritma yang dicadangkan telah dibangunkan di dalam persekitaran simulasi grid yang dikenali sebagai GridSim dan dinilai dengan algoritma toleransi sesar yang lain seperti ACO berdasarkan kepercayaan, ACO toleransi sesar, ACO tanpa toleransi sesar, dan ACO dengan toleransi sesar dari segi masa pelaksanaan keseluruhan, purata latensi, purata masa pelaksanaan, daya pemprosesan, kadar kejayaan pelaksanaan, dan pengimbangan beban. Keputusan eksperimen menunjukkan algoritma yang dicadang berjaya mencapai prestasi yang baik dalam kebanyakan aspek, dan kedua terbaik dari segi pengimbangan beban. DAFTS telah mencapai kenaikan yang terendah pada masa pelaksanaan, purata pelaksanaan dan purata latensi masing-masing sebanyak 7%, 11% dan 5%, dan penurunan daya pemprosesan dan kadar kejayaan yang terendah masing-masing sebanyak 6.49% dan 9% apabila kadar kegagalan semakin meningkat. DAFTS juga telah mencapai kadar kenaikan yang paling rendah pada masa pelaksanaan, purata masa pelaksanaan dan purata latensi masing-masing sebanyak 5.8, 8.5 dan 8.7 kali, dan kenaikan yang tertinggi pada daya pemprosesan dan kadar kejayaan tertinggi masing-masing sebanyak 72.9% dan 93.7% apabila bilangan kerja semakin bertambah. Algoritma yang dicadangkan dapat menyelesaikan masalah pengimbangan beban secara lebih efektif dan meningkatkan kadar kejayaan pelaksanaan di dalam sistem teragih yang terdedah kepada kegagalan.

Kata Kunci: Penjadualan grid, Toleransi sesar, Pemprosesan semula kerja, Titik semak kerja, Sistem koloni semut.

Abstract

Fault tolerance in grid computing allows the system to continue operate despite occurrence of failure. Most fault tolerance algorithms focus on fault handling techniques such as task reprocessing, checkpointing, task replication, penalty, and task migration. Ant colony system (ACS), a variant of ant colony optimization (ACO), is one of the promising algorithms for fault tolerance due to its ability to adapt to both static and dynamic combinatorial optimization problems. However, ACS algorithm does not consider the resource fitness during task scheduling which leads to poor load balancing and lower execution success rate. This research proposes dynamic ACS fault tolerance with suspension (DAFTS) in grid computing that focuses on providing effective fault tolerance techniques to improve the execution success rate and load balancing. The proposed algorithm consists of dynamic evaporation rate, resource fitness-based scheduling process, enhanced pheromone update with trust factor and suspension, and checkpoint-based task reprocessing. The research framework consists of four phases which are identifying fault tolerance techniques, enhancing resource assignment and job scheduling, improving fault tolerance algorithm and, evaluating the performance of the proposed algorithm. The proposed algorithm was developed in a simulated grid environment called GridSim and evaluated against other fault tolerance algorithms such as trust-based ACO, fault tolerance ACO, ACO without fault tolerance and ACO with fault tolerance in terms of total execution time, average latency, average makespan, throughput, execution success rate and load balancing. Experimental results showed that the proposed algorithm achieved the best performance in most aspects, and second best in terms of load balancing. The DAFTS achieved the smallest increase on execution time, average makespan and average latency by 7%, 11% and 5% respectively, and smallest decrease on throughput and execution success rate by 6.49% and 9% respectively as the failure rate increases. The DAFTS also achieved the smallest increment on execution time, average makespan and average latency by 5.8, 8.5 and 8.7 times respectively, and highest increase on throughput and highest execution success rate by 72.9% and 93.7% respectively as the number of jobs increases. The proposed algorithm can effectively overcome load balancing problems and increase execution success rates in distributed systems that are prone to faults.

Keywords: Grid computing, Scheduling optimization, Fault tolerance, Load balancing, Ant colony system.

Acknowledgement

Alhamdulillah, with all the blessing from Allah and invaluable contribution from countless people, this thesis is finally completed.

Firstly, I would like to express my sincere gratitude to my main supervisor, Prof. Dr. Ku Ruhana Binti Ku Mahamud, for all the guidance and support since the beginning of this journey. Being supervised by her will become one of the best experiences in my life. I would like to also appreciate my second supervisor, Prof. Dr. Hiroaki Morino, who has provided ideas and thoughts, and treated me with hospitality during exchange programs in Tokyo. For sure, I am so indebted to them.

Secondly, thanks to the love of my life, Dr. Husna Binti Jamal Abdul Nasir, who has been supporting me physically and mentally throughout this journey. Without her, I do not think I will be able to achieve this. I would like to also express special dedication to my beloved kids, Danish Adni, Daniel Adib and Daris Adel for being so patience and understanding. Not forgetting my late father, Bukhari Bin Husin who returned to The Creator on August 17, 2020, and my beloved mother, Saudah Binti Ahmad, without them, I will never be who I am right now.

Finally, I would like to express my appreciation to parents in law, siblings, relatives, friends, co-workers and managers in Intel Microelectronics that have supported me directly or indirectly. I apologize that I am unable to name every single person, but certainly each of them deserves my utmost appreciation.

Thank you.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

B/S	Bytes per second
GIS	Grid information service
MI	Million instructions
MIPS	Million instructions per second
NP	Non polynomial
PE	Processing element
PV	Pheromone value

CHAPTER ONE

INTRODUCTION

Grid computing emerged from meta-computing in the mid-1990s with the introduction of middleware to serve as a wide area infrastructure to support diverse online processing and data intensive applications (Foster & Kesselman, 2004; Moallem, 2009; Sotiriadis, Bessis, Xhafa, & Antonopoulos, 2012; Wang, Jie, & Chen, 2018). During that time, several systems were developed to support scientific applications such as Globus Toolkit (Foster & Kesselman, 1997; Severance, 2014), Storage Resource Broker (Baru, Moore, Rajasekar, & Wan, 1998; Hsu et al., 2014), Legion (Grimshaw, Ferrari, Knabe & Humprey, 1999; Rubab, Hassan, Mahmood & Shah, 2015) and Condor-G (Frey, Tannenbaum, Livny, Foster & Tuecke, 2002; Ashraf & Mazher, 2013). To further improve the functionality and standardization of grid computing technology, the Global Grid Forum was established in 1998 as an international community and standards organization which is responsible for controlling the standards to be developed and to run multiple standardization activities (Moallem, 2009). Later, in 2002, the Open Grid Services Architecture was officially established as a standard community that developed the Globus Toolkit 3.0 and 3.2 based on the Open Grid Services Infrastructure and, most recently, introduction of the Globus Toolkit 4.0 (Talia, 2002; Kim, Kim & Weissman, 2014).

Grid computing is the collection of computer resources located in different locations that work together to complete assigned tasks. Grid computing has been widely used in solving challenging problems in real world situations such as video analysis (Zorrilla et al., 2017), protein folding (Natrajan et al., 2004; Dill & MacCallum, 2014), hydrology modelling (Lecca et al., 2011), natural disasters simulation (Pajorova & Hluchý, 2012; Yuan, 2016) and bioinformatics (Merelli, 2019). The main reasons for deploying grid computing are to introduce a system that is scalable, simple to use, autonomic and able to deal with faults (Qureshi, Khan, Manuel & Nazir, 2011).

Grid computing can be further classified into data grid, service grid and computational grid (Azeez & Haque, 2011; Muthu & Kumar, 2017). A data grid is mainly used for storing large data sets which will be segmented and stored in different storage locations (Bansod, Virk, & Raval, 2018). A service grid is generally used for maintaining the services, analyzing resources, scheduling tasks and security (Madi, Yusof, Tahir, Zaini & Hassan, 2017). Last, the computational grid, which consists of a highly distributed environment and dynamic in nature, uses collective resources to solve a single computational problem (Shah, Mahmood, Rubab & Hassan, 2016). The computational grid, which is based on dynamically distributed resources and large scale sharing, presents a tremendous amount of low cost computational power (Yan, Wang, Wang & Chang, 2009; Patel & Sharma, 2019). In other words, maximum computational power can be achieved with minimal cost to execute heavily loaded tasks in addition to the reliability and efficiency without the need for dedicated resources. Due to the heterogeneous nature of computing resources within the grid computing environment, effective resource management needs to be present in order to ensure maximum utilization of grid computing capabilities.

The grid computing system consists of several main components which are user interface, security, workload management, scheduler, data management and resource management. The user interface acts as a virtual wall between complexities of the grid computing system and end users. When it involves user interaction, security should be involved to support authentication, authorization, data encryption and data validation. Jobs and resources that are managed by workload management are passive entities which will not run unless being instructed. To start the process of giving instruction to either entity, the broker service needs to identify jobs and resources before execution. Then, the scheduler will schedule jobs and resources for execution. The next step, which is to facilitate the execution by assigning jobs to suitable resources, validates the status and retrieves the results which, after completion, will be handled by the job and resource management component. Finally, the actual data transfer from one destination to another will be managed by the data management component (Bienkowski, 2018).

Job and resource management are part of the main components that need to be considered in order to administer all submitted jobs and available resources (Venkatesan, Ramalakshmi, & Latha, 2018). Unlike resource management in traditional computing systems where resource managers have full control of a resource, grid resource management is focused on managing and provisioning independently owned and administered resources; this is very complicated due to the heterogeneity of each resource (Foster & Kesselman, 2004; Patel & Sharma; 2018). There are various issues in grid resource management such as resource discovery, resource scheduling, resource monitoring, resources inventory, resource provisioning, load balancing, fault tolerance, autonomic capabilities and service level management systems (Li, Xie, Qi, Luo, & Xie, 2011; Idris, Ezugwu, Junaidu & Adewumi, 2017; Darmawan & Aradea, 2019). Job scheduling focuses on applying effective scheduling decision based on defined parameters such as required computation power, time to complete computation, current load and capacity, size of job with main objectives to minimize execution time and maximize throughput (Yadav, Jindal & Singh, 2013). Job scheduling is also directly related to load balancing as it ensures fair jobs distribution among available resources, maximizes throughput, reduces latency and avoids stagnation (Patni, Aswal, Agarwal & Rastogi, 2015). Fault tolerance is crucial to be considered since, in distributed systems, specifically grid computing, faults are unavoidable due to the heterogeneous nature of resources that might have different fitness and reliability (Souli-Jbali, Hidri, & Ayed, 2019; Khaldi, Rebbah, Meftah & Debakla, 2020).

Job scheduling can be further classified as static and dynamic scheduling (Moallem, 2009; Balasangameshwara & Raju, 2012). In static scheduling, jobs and resources assignment is done before the execution begins by using all the known information and the whole execution will be based on pre-defined parameters. However, this is not the case in a real grid computing system because parameters may change from time to time based on previous execution results, occurrence of failures, stability of the environment and priority of tasks. This is where dynamic scheduling plays an important role in making scheduling decisions during runtime. Thus, jobs and resources assignment will be completed efficiently by considering the most up-to-date information during runtime which may result in better performance as compared to static scheduling.

Load balancing is to improve the distribution of jobs to available resources to maximize resource utilization. Few criteria are being used to determine the load of resources which include load of processor, latency, communication overhead, memory usage and node priority (Vaghela, 2014). There are several common steps in load balancing algorithms which include load monitoring, synchronization, rebalancing criteria and job migration (Rathore & Chana, 2014). These steps also exist in fault tolerance algorithms to mitigate resource failures by migrating failed jobs to alternative resources. Thus, it is important to consider the load balancing aspect as well in the fault tolerance to ensure that the system can run optimally since both are mutually inclusive.

In typical distributed computing systems that involve parallel computation such as grid, cluster and cloud computing, there are many shared resources to process submitted jobs; it is, therefore, common for failure to happen during job processing. Many types of failure can occur, such as network failure, packet loss and corruption, physical failure to the central processing unit, hard drive and storage drive in the processing machine, user termination, service and protocol failure, software failure and processing failure (Rakheja, Kaur & Rkheja, 2014; Savyanavar & Ghorpade, 2019). Out of the most common types of failure, it is possible to resolve network and processing failures in real time through utilization of a proper fault tolerance strategy. If failure happens in the processing machine, users will experience delay in execution time (Amoon, 2012; Aliyu, Mohammed, Abdulmumin, Adamu & Jauro, 2020). This is because submitted jobs cannot be processed effectively and resources will not be released to process subsequent jobs in the queue. As a result, stagnation will occur where the throughput will be greatly decreased or totally stalled due to limited resources available to process jobs in the queue. Therefore, fault tolerance is the main requirement of distributed system (Krasovec & Filipcic, 2019).

Generally, fault tolerance can be categorized as static and dynamic (Xu, Cai, He & Tang, 2019). Static fault prevention assumes that all the information, including the jobs' and resources' characteristics, and success or failure state, are known in advance. However, in a distributed computing environment such as a grid, this technique is not relevant because the environment itself is dynamic in nature (Balasangameshwara & Raju, 2012; Liu & Guo, 2019). The most suitable technique is dynamic fault prevention which relies on run time state information to make the decision. This means that the fault prevention mechanism will not be executed until the fault is detected. Furthermore, the type of fault will also be considered in order to decide whether to reprocess the job using the same resource or migrate to another resource.

The most common fault tolerance techniques consist of checkpoint recovery and job replication (Garg & Singh, 2011; Altameem, 2013; Bougeret, Casanova, Robert, Vivien & Zaidouni, 2014; Ebenezer, Rajsingh, & Kaliaperumal, 2019). Checkpoint recovery relies on the record of the last saved state which is stored temporarily and can be a reprocess point in the presence of failure. The job reprocessing does not need to be restarted from the beginning, it can start at the last saved state. This approach gives significant time saving to reprocess failed jobs (Rathore, 2017; Garba et al., 2020). Another technique is job replication which is the action to submit duplicate jobs to multiple resources with the assumption that if one execution fails, the results of execution from another resource can be used. However, this approach requires very high technical considerations because it may potentially overload the entire system (Singh, 2016).

There are also other fault tolerance techniques such as job migration (Qureshi, Khan, Manuel & Nazir, 2011; Prashar, Nancy & Kumar, 2014), job retry (Wenming, Zhenrong & Peizhi, 2009; Rathore & Chana, 2015; Idris et al., 2017) and penalty (Keerthika & Kasthuri, 2011; Sharma, Sharma & Dalal, 2014; Kurochkin & Gerk, 2018). Job migration is essential in dynamic scheduling as it allows a failed job to be submitted to other resources to reduce the possibility of another failure and allow the failed resource to recover. Job retry is performed by submitting the original job or the last saved job to the execution queue to undergo the standard scheduling process. Job retry is considered as critical as it ensures the failed job can be reprocessed until completion. Penalty is a technique used to penalize the occurrence of failure, either to the resource, or to the path that leads to the resource so that they become less desirable during the job scheduling process.

Fault tolerance has been widely implemented in various distributed computing systems (Kumar & Pathak, 2018; Chinnathambi, Santhanam, Rajarathinam & Senthilkumar, 2019). The resilient distributed dataset is one example of a fault tolerance strategy implemented in cluster computing applications. The general framework proposed is to log the transformations used to build a dataset which provides enough information for a quick recovery process in case of partition loss (Zaharia et al., 2012). A fault tolerance load balancing algorithm, proposed by Balasangameshwara and Raju (2012), is an example of fault tolerance application in grid computing which first backs up the job before discovering potential resources to process it. Then, the fault manager will detect or monitor the fault and apply a rescheduling policy for a job stored in the primary backup to another resource in the presence of fault.

Fault tolerance, scheduling and load balancing are defined as Nondeterministic Polynomial (NP)-complete problem (Glaßer, Pavan & Travers, 2009) which means that there is no exact algorithm that can solve them in a polynomial time (Blum & Roli, 2003; Pooranian, Shojafar, Abawajy & Singhal, 2013). Table 1.1 shows different types of NP-complete problems that are grouped by the type of problem such as scheduling, routing, assignment, subset problems and others. In grid computing specifically, several types of NP-complete problems such as job scheduling, load balancing and fault tolerance are to be tackled together to improve the system's functionality and performance.

Table 1.1

Type of NP-complete problems

NP-Complete Problem				
Routing	Scheduling	Assignment	Subset	Others
Network Routing	Job Scheduling	Course Timetabling	Multiple Knapsack	Load Balancing
Travelling Salesman Problem	Project Scheduling	Quadramatic Algorithm	Set Covering	Constraint Satisfaction
Sequential Ordering	Total Weighted Tardiness	Graph Coloring		Digital Image Habitats
Vehicle Routing	Flow Shop			Protein Folding
Query Routing				Fault tolerance
				Bus Stop

	Bin Packing and Cutting Stocks

Note. Adapted from Nasir (2020).

The most popular way to solve these problems is to use approximate or metaheuristic algorithms such as GA (Werner, 2011; Sajedi & Rabiee, 2014; Kapil, Chawla, & Ansari, 2016; Younis & Yang, 2018), simulated annealing (Lin & Vincent, 2012; Vincent, Redi, Hidayat, & Wibowo, 2017), Tabu Search (TS) (Kong, Shen, Chen, Wang & Song, 2010; Glover & Laguna, 2013; Lai, Demirag, & Leung, 2016) and, recently, ACO (Ku-Mahamud & Alobaedy, 2012; Martin, Cervantes, Saez, & Isasi, 2020). GA, SA, TS and ACO are some of the local search algorithms used to search a solution space by moving one solution to another and constructing the best solution for scheduling and the load balancing algorithm. A feasible solution is quickly produced by using these methods, but it will not come close to the optimal solution. The solution produced by one metaheuristic algorithm can also be improved by applying other metaheuristic or non-heuristic algorithms to obtain a better solution.

1.1 Background

Job scheduling is an important process in grid computing to effectively identify suitable resources to process jobs submitted by the user. Job scheduling is classified into two categories which are static scheduling and dynamic scheduling (Yadav, Jindal & Singh, 2013). In static scheduling, the resource assignment is performed before the execution while in dynamic scheduling, the scheduling decision can be performed during execution. The dynamic scheduling scheme performs prediction based on historical records (Kaur & Aggarwal, 2013). It is often enhanced to provide fault tolerance capability as it allows the system to adapt with dynamic environment as well as making scheduling decision during execution.

Fault tolerance in the grid consists of three main strategies: fault detection or identification, fault prediction and fault recovery. Fault detection or identification generally means detecting the type of fault when it occurs before mitigating it with the most suitable solution. On the other hand, fault prediction entails predicting the probability of faults occurring based on historical data and applying a suitable scheduling policy to reduce fault probability. Last but not least, fault recovery consists of several popular techniques such as job replication (space-sharing) and checkpointing (time-sharing) (Altameem, 2013). The advantage of job replication is that it does not require re-computation because each job has several simultaneous copies assigned to different resources; therefore, if one fails, the other can still be processed (Vansa, 2019). However, this technique is not very effective because a copy of a job is considered as an individual execution and may potentially congest the job queue. Another technique is checkpointing which requires the state of the running task to be stored at defined checkpoints and if the job fails the execution will restart from the last saved state instead of from the beginning. However, the drawback is that having too many checkpoints may lead to runtime overheads (Idris et al., 2017; Garba et al., 2020).

In addition to fault tolerance, load balancing is also important since the majority of fault tolerance algorithms focus on fault strategies rather than load balancing. Load balancing is categorized into two types: static and dynamic. In static load balancing, information about jobs and resources is known prior to initialization and the scheduling results are obtained even before all the jobs are executed (Prajapati, Rathod & Khanna, 2015). On the other hand, dynamic load balancing is preferred since it uses runtime state information to make decisions and its decentralized parameters provide better scalability and fault tolerance (Sharma & Dalal, 2014). Load balancing can also be incorporated with job migration in order to solve load balancing problems and provide fault tolerance by using the checkpoint technique (Rathore & Chana, 2015). Without proper load balancing, stagnation may occur because the computational time of the processed job is high. Stagnation may also occur when all jobs are assigned to the same resources which, consequently, leads to the resources having high workloads. Thus, it is critical to effectively utilize all resources to minimize stagnation problems in grid computing.

Figure 1.1 depicts the general flow of job scheduling, fault tolerance and load balancing. When jobs are submitted, job scheduling will take place to determine the suitable resources to accept the jobs. During this process, indirectly the load balancing is already being considered as the resource load is checked before assigning the jobs. During execution, fault tolerance process will be invoked upon failure and the job scheduling will be re-invoked to perform another round of scheduling to assign failed jobs to alternative resources. It can be concluded that job scheduling, load balancing and fault tolerance are mutually inclusive in heterogenous and dynamic nature of grid environment (Balasangameshwara, 2014).

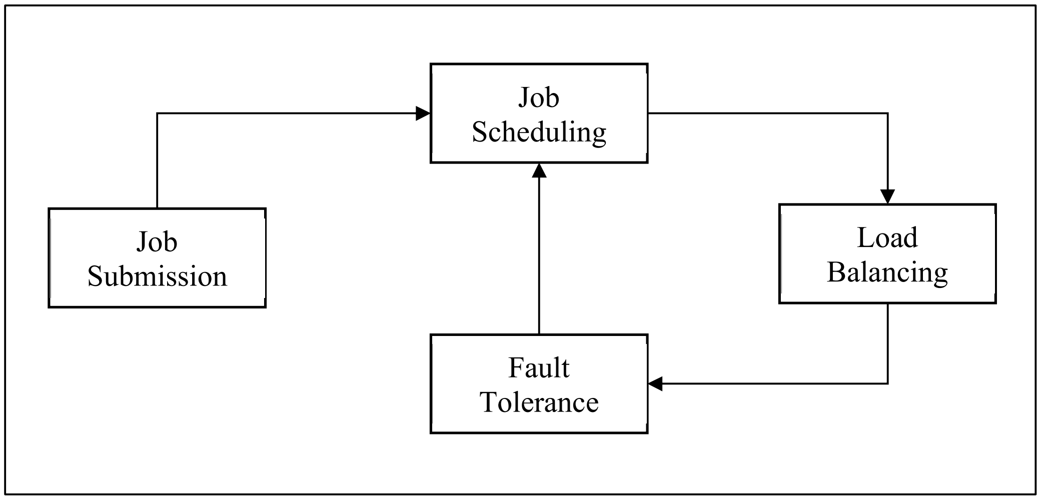


Figure 1.1. General flow of job scheduling, load balancing and fault tolerance

The ACO algorithm is used because it can be easily adapted to solve both static and dynamic combinatorial optimization problems (Lorpunmanee, Sap, Abdullah & Chompoo-inwai, 2007; Ku-Mahamud & Alobaedy, 2012; Goyal & Singh, 2012; Ankita & Sahana, 2019) because it is designed to find unknown optimal solution where the pheromone values are associated with solution components (Blum, 2005). ACO is flexible and can be modified and combined with other nature inspired swarm intelligence approaches such as Intelligent Water Drop in order to speed up optimal scheduling in addition to minimizing makespan, balancing the load and utilizing resources efficiently (Mathiyalagan, Sivanamdam & Saranya, 2013). Another study that combined ACO with other algorithms was proposed by Modiri, Analoui and Jabbehdari (2011) where their proposed algorithm combines the ACO algorithm and Directed Acyclic Graph (DAG) method in order to cater both for load balancing and fault tolerance aspects.

There are many variations of the ACO algorithm such as Ant System, Ant Colony System (ACS), Max-Min Ant System (MMAS), Rank-based Ant System and Elitist Ant System (Dorigo & Stützle, 2004). Ant colony optimization has been successfully applied to solve many routing problems such as the network routing problem (Ye & Mohamadian, 2014; Yang, Ping, Aijaz, & Aghvami, 2018), vehicle routing problem (Tan, Lee, Majid, & Seow, 2012; Kuo & Zulvia, 2017), travelling salesman problem (Holzinger et al.; 2016; Gülcü, Mahi, Baykan, & Kodaz, 2018), sequential ordering routing problem (Gambardella, Montemanni, & Weyland, 2012; Ezzat, 2013; Skinderowicz, 2017) and query routing problem (Santillán, Reyes, Conde, Schaeffer & Valdez, 2010; Hanane & Fouzia, 2014). ACO has also been successfully applied in fault tolerance (Idris et al., 2017) which has resulted in better load balancing in the presence of failure.

The ACS is considered as one of most widely used ACO variants for solving NP-complete problems (Schyns, 2015; Nasir, Ku-Mahamud, & Kamioka, 2017; Liu et al., 2018). The ACS consists of two main mechanisms: exploration and exploitation. Exploration relies on the transition probability between nodes while exploitation chooses the node with the highest pheromone to reduce calculation time as well as ensure reliable path selection. In addition to both mechanisms, local pheromone update is also introduced in the ACS on top of the global pheromone update to increase pheromone intensity of the best solution so far in order for the exploitation mechanism to work. Thus, it is important to balance between exploration and exploitation so that the solution is not too biased or not too random to provide the most optimal solution.

This research aims to enhance ACS algorithm for dynamic fault tolerance in grid computing to overcome both fault and load balancing problems. The proposed algorithm extended the ACS scheduling algorithm combined with the checkpoint and suspension techniques to provide efficient scheduling in faulty environments. Checkpointing was adapted and adopted from Prashar, Nancy and Kumar (2014) in order to save the state of execution based on certain intervals so that reprocessing can start from the last saved state instead of from the beginning. The suspension technique was inspired from the trust mechanism proposed by Wenming et al. (2009) in which resources are rewarded or penalized based on execution status. In this research, the trust mechanism is further combined with the suspension technique because it is possible that a recently failed resource may still have high levels of pheromone that will cause it to be reassigned without undergoing the recovery process.

1.2 Problem Statement

The process of identifying resource failure or faults in dynamic grid computing is complicated due to its distributed and heterogeneous nature (Ku-Mahamud, Din & Nasir, 2011; Haider & Nazir, 2016). In typical fault tolerance algorithms, the scheduling and rescheduling process often considers the resource load when assigning jobs but not the execution history or fitness of resources. This may lead to uncertain success rate as resources with low load could have high possibility of failure. For instance, Idris et al. (2017) proposed an improved ACO algorithm with fault tolerance in the grid by considering the resource load when assigning jobs to minimize processing time and increase throughput. However, resource load alone does not indicate the fitness of the resource which could lead to lower execution success rate.

Load balancing is often considered during scheduling process to fairly control the distribution of jobs to available resources to maximize resource utilization (Khan, 2017; Sheikh, Nagaraju & Shahid, 2018). However, typical fault tolerance algorithms do not possess effective the load balancing technique due to main objectives to maximize success rate and throughput (Idris et al., 2017). There are also algorithms that have considered the load balancing aspect but disregarded the execution success rate such as algorithms proposed by Prashar et al. (2014), Rajab and Kabalan (2016) and Garba et al. (2020). Without considering the load balancing in faulty environment, resource utilization will be poor and may lead to stagnation as some resources will be overloaded with high number of jobs. For example, fault tolerance algorithm proposed by Prashar et al. (2014) applied effective mitigation technique called resubmission based on checkpoint but did not consider the load balancing aspect. In addition, Garba et al. (2020) proposed a fault tolerance algorithm that dynamically controlled the checkpoint interval to improve makespan, throughput and turnaround time but did not consider the load balancing aspect.

Temporary resource isolation in the presence of failure is essential to penalize and suspend resources that failed to complete execution (Wenming et al., 2009). This aspect has not been the main focus in fault tolerance algorithms as the main objective is to reprocess failed jobs to alternative resources. As a result, resources that recently failed to complete execution may still be assigned with majority of jobs and eventually lead to higher possibility of another failure. Thus, it is important to temporarily isolate recently failed resources so that it can undergo recovery process and complete the remaining jobs in the queue. One example is tentative ACO algorithm which was proposed by Sharma et al. (2014) that considered load balancing by applying an encouragement and punishment argument based on execution status but did not include the strategy to reprocess the failed job.

Due to all these limitations, improvement of ACS-based fault tolerance algorithms is essential in order to extend the capability of both fault tolerance and load balancing aspects to effectively overcome load balancing problems, minimize execution time and latency, and maximize throughput and execution success rates in the presence of failures. This leads to several research questions as follows:

1. What are the effective fault tolerance techniques that can be enhanced to consider the load balancing aspects?
2. How can the resource fitness and trust factors be considered in the fault tolerance management?
3. How can the ant-based fault tolerance algorithm be improved to provide fault tolerance and load balancing aspects?
4. How effective is the proposed algorithm to cater for fault tolerance and load balancing in grid environments?

1.3 Research Objectives

The main objective of this research is to develop an enhanced ACS-based algorithm for dynamic fault management in grid computing which can assign jobs to suitable resources, identify failures and ways to mitigate them, resubmit jobs to other available resources whenever required, apply penalties and suspend failed resources temporarily to avoid being assigned to the next execution cycle, overcome stagnation, minimize computational time and balance the load of entire resources in the grid environment on every execution.

Specific objectives of the research are:

1. To investigate fault tolerance techniques that can be enhanced in the context of load balancing.
2. To develop an ant-based fault tolerance algorithm that considers resource fitness, apply temporary suspension, and control pheromone assigned to resources.
3. To develop an improved ant-based fault tolerance algorithm that considers both fault tolerance and load balancing aspects.
4. To evaluate the improved ant-based fault tolerance algorithm in simulated grid computing environment.

1.4 Significance of the Research

Grid computing is emerging as a new computing paradigm to solve challenging applications in engineering, science and economics. As a consequence, grid architecture has to consider managing distributed, heterogeneous and dynamically available resources in efficient ways. Therefore, the management of resources and failures is crucial in grid computing environments so that every component of the whole execution process works flawlessly (Azeez & Haque, 2011; Rathore & Chana, 2014).

The outcome of this research contributes to a new variant of ACO algorithm with fault tolerance capability using checkpoint-based job resubmission to other resources. This capability is to ensure that the failed job will be executed completely from the last saved state (Prashar, Nancy & Kumar, 2014). In addition to that, the resource suspension technique which was inspired from penalty application allows a recently failed resource to recover (Wenming et al., 2009). The aspect of load balancing and scheduling are improved which would enhance the basic approach of the ACO algorithm by dynamically reducing faults using effective and reliable method. At the same time, it tries to balance the load of entire resources to ensure fair distribution and execution of jobs to overcome stagnation in the presence of faults. Eventually, it is possible to implement the proposed algorithm in other types of distributed computing system such as cloud computing, wireless sensor networks and cluster computing.

1.5 Scope and Limitations of the Research

This study focuses on improvement of the ACO algorithm by developing a new fault tolerance algorithm called Dynamic Ant Colony System-based Fault Tolerance with Suspension (DAFTS) that can overcome stagnation problems and offer fault tolerance measures in grid computing. The focus is on improving the way ants search the best resources in processing jobs and identify failures by using a fault detection approach and, at the same time, trying to dynamically apply fault recovery techniques. The proposed algorithm is based on the Ant Colony System (ACS) algorithm to improve the scheduling and pheromone update for fault prevention measures and the load balancing process. The dynamic fault tolerance workflow was used to experiment in a simulation environment called GridSim by using both dynamic and static characteristics to simulate various scenarios and conditions.

The limitation of the research is the experiments were done in simulated grid computing environment called GridSim which is designed to provide close to actual functionality as the real grid computing environment. In real environment, the DAFTS might not behave the same way as in simulated environment due to unknown factors but the difference in its behavior will not be so significance. Another limitation of the research is the DAFTS is based on single ACO and is experimented by defining parameters for each batch of execution. Each execution represents one session in which the grid users submitted the jobs in real grid environment.

Even though grid computing has been available for many years, improvement can be further explored in order to make it relevant to the constantly changing computing world. Fault tolerance in grids is the main improvement to be explored in this study in addition to improving load balancing. Looking at the concept of the load balancing algorithm, it is definitely possible to also use the same concept to handle failure either by introducing a separate algorithm just to handle failure or integrate both fault tolerance and the load balancing algorithm to simplify the architecture as well as to allow the new algorithm to be implemented in existing load balancing algorithms applied in real applications. ACO has been selected since it gives flexibility to combine with other algorithms and is proven to be one of the most reliable algorithms to handle load balancing in grid computing.

1.6 Structure of the Thesis

The structure of this thesis is as follows. In Chapter 2, an overview of grid computing that covers job scheduling and load balancing, fault tolerance in grid computing and

ACO that covers scheduling, load balancing and fault tolerance are discussed. In addition, issues and limitations are also elaborated upon.

Chapter 3 covers the framework and methodology used to realize the objectives of DAFTS in grid computing, followed by the simulation model, simulation design and evaluation methodology and performance evaluation metrics

Chapter 4 presents, in detail, the proposed DAFTS algorithm which includes the load balancing technique using dynamic scheduling with checkpoint and resource suspension. Furthermore, the pseudocode of the DAFTS is also presented.

Experimental design and parameters tuning of the DAFTS algorithm are covered in Chapter 5. Then, the performance of the proposed algorithm is compared with the other four algorithms by measuring the effect of different failure rates and different numbers of tasks in terms of execution time, throughput, makespan, latency, load balancing and execution success rate.

Chapter 6 is dedicated for detailed discussion on Chapter 3, Chapter 4 and Chapter 5 and relationship between key contributions of each chapter. Finally, Chapter 7 discusses the contributions of the research and highlights future research directions.

CHAPTER TWO

LITERATURE REVIEW

This chapter presents the review of research that has been carried out in the areas of grid computing and ACO. The overview of grid computing is discussed in Section 2.1 which covers previous works related to grid scheduling and load balancing. Section 2.2 further elaborates fault tolerance, including techniques that are applied in the grid environment. Section 2.3 explains in detail about ACO and its applications such as scheduling, load balancing and fault tolerance in grid computing, followed by Section 2.4 that summarizes this chapter.

2.1 Grid Computing

Grid computing is a high performance computational system that consists of decentralized distributed resources connected by a network that offers cost effective high performance computing capability and allows a more cooperation and collaboration between resources to achieve common objectives. Grid computing is deployed within the standard Internet protocol architecture, specifically at the application, transport, Internet and link layers (Foster, Kesselman & Tuecke, 2001; Hwang, Dongarra & Fox, 2012; Basu, 2016).

As illustrated in Figure 2.1, majority of grid computing layers within the grid protocol architecture such as collective layer, resource layer and connectivity layer are located at the same level as application layer in Internet protocol architecture (Nassiry & Kardan, 2009; Kamra & Chugh, 2011). Collective layer deals with collaborative operations between shareable resources while resource layer covers actions related to network parts such as negotiation, initiation and monitoring. All the associated protocols, communication and authentication are placed under connectivity layer. Last but not least is the fabric layer in which all shareable resources are placed (Seelwal, 2014; Ankita & Shana, 2018).

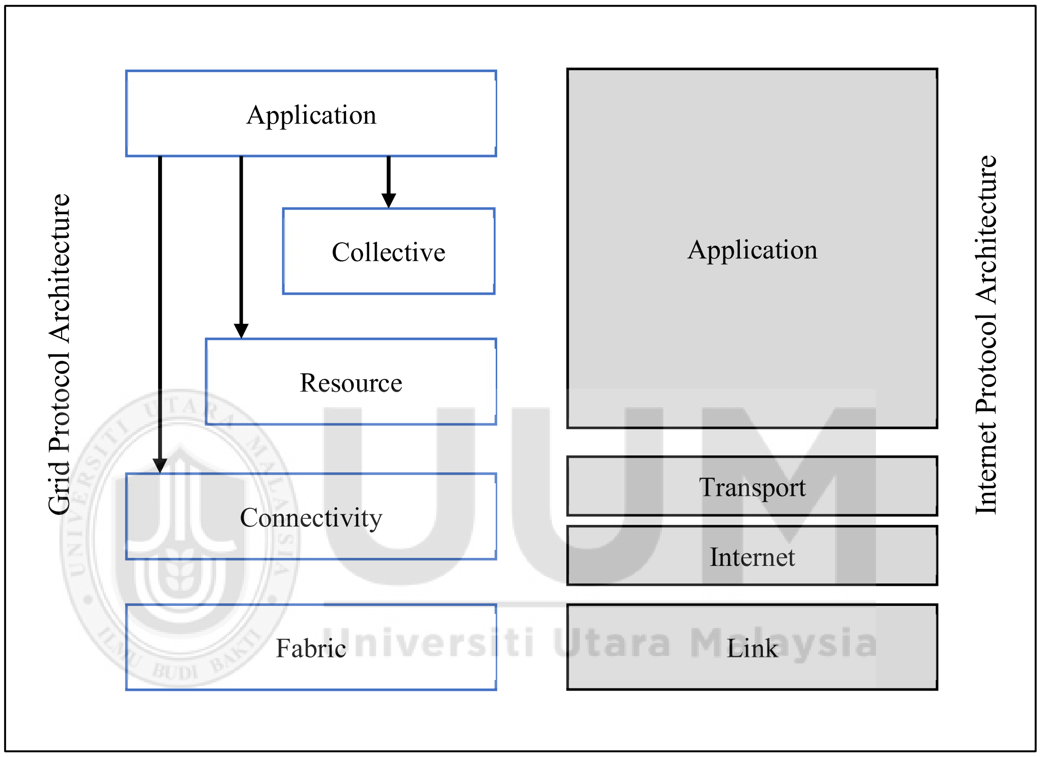


Figure 2.1. Grid protocol architecture vs. Internet protocol architecture

Resources in grid computing are combined from different locations to form a super virtual computer to solve large and complex tasks (Levitin, Xing, Johnson & Dai, 2018). Individual users can access several computing resources such as data, applications, storage, and processors without knowing the locations of each resource. The grid resource broker is a key element in grid computing that is responsible for identifying and matching suitable resources from multiple administrative domains to process a submitted job (Kaushik & Vidyarthi, 2018). As shown in Figure 2.2, the grid resource broker is connected to the users and resources. It is responsible for managing scheduling, load balancing, and faults.

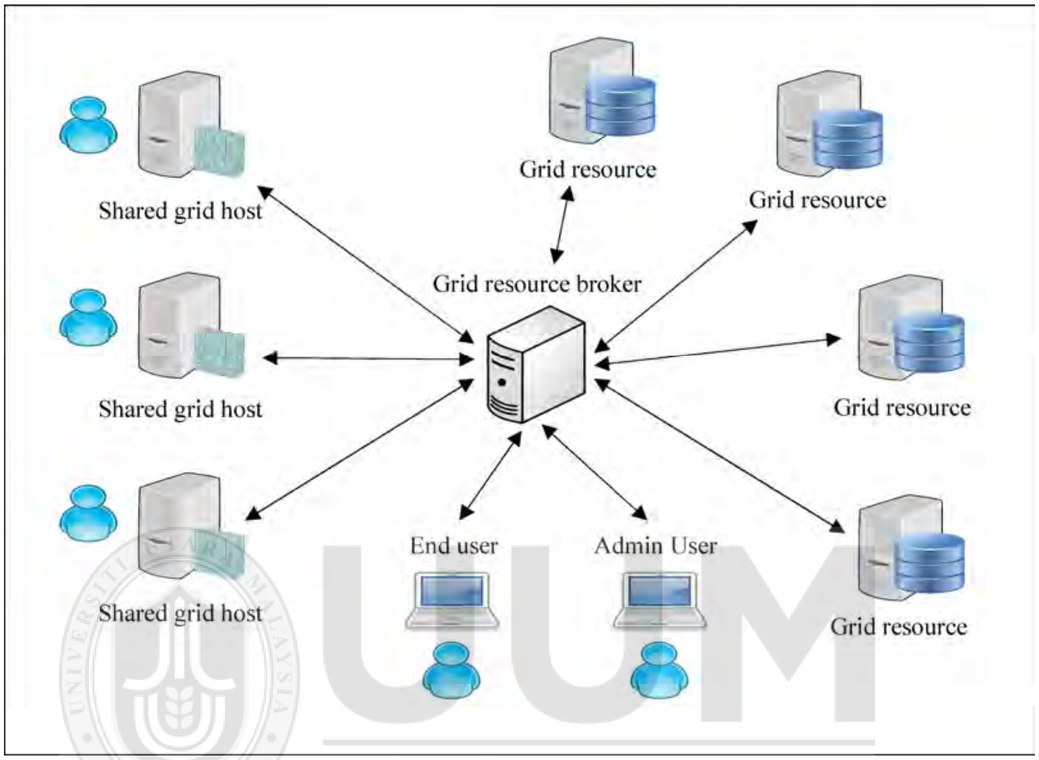


Figure 2.2. High level architecture diagram of grid system

Resources in grid computing systems are not under central control, where they can enter and leave the system at any time (Meo, Messina, Domenico & Sarné, 2015). An effective resource management system is needed to manage the grid computing system. Resource management is a central component of a grid computing system and is responsible for managing submitted jobs and available grid resources which includes resources allocation, assignment, authorization, assurance and authentication to process submitted jobs (Sharma & Bawa, 2008; Qureshi et al., 2014).

Main problems being solved in grid computing are directly aligned with problems in distributed systems where multiple computing capabilities are connected through networks to perform coordinated computations. These problems include scheduling (Jiang & Chen, 2015; Rathore, 2015; Alkhanak, Lee, Rezaei & Parizi, 2016; Maipan-uku, Konjaang & Baba, 2016; Sathish & Reddy, 2017; Younis & Yang, 2017; Mahato, Sandhu, Singh & Kaushal, 2019; Eng, Muhammed, Mohamed & Hasan, 2020), load balancing (El-Zoghdy & Alaa, 2015; Patni & Aswal, 2015; Naik, Jagan & Narayana, 2015; Rathore, 2015; Idris et al., 2017; Omer & Abdalla, 2018; Mahato et al., 2019; Garba et al., 2020), resource allocation (Satish & Reddy, 2018; Krasovec & Filipcic, 2019; Shukla, Kumar & Singh, 2018) and fault tolerance (Singh & Bawa, 2016; Abdullah, Ali & Haikal, 2017; Haider & Nazir, 2017; Idris et al., 2017; Goswami & Das, 2018; Ahuja & Banga, 2019; Garba et al., 2020. A good grid management system should consider scheduling, load balancing and fault tolerance to ensure the optimization and efficiency that are important in a dynamic environment that performs intensive computations.

2.1.1 Grid Computing versus Cloud Computing

Grid computing and cloud computing are not a completely new concept. While cloud computing is becoming a popular paradigm, it does not overtake the importance of grid computing paradigm. Generally, cloud computing is specialized in scalability, services, economic and configurability (Sharma, 2013). Cloud computing overlaps with many distributed computing technologies such as grid computing and cluster computing. Cloud is built within the web service architecture where it is more focused on service orientation applications. Figure 2.3 clearly illustrates the overlapping of grid computing and cloud computing within the distributed systems (Foster, Zhao, Raicu & Lu, 2008).

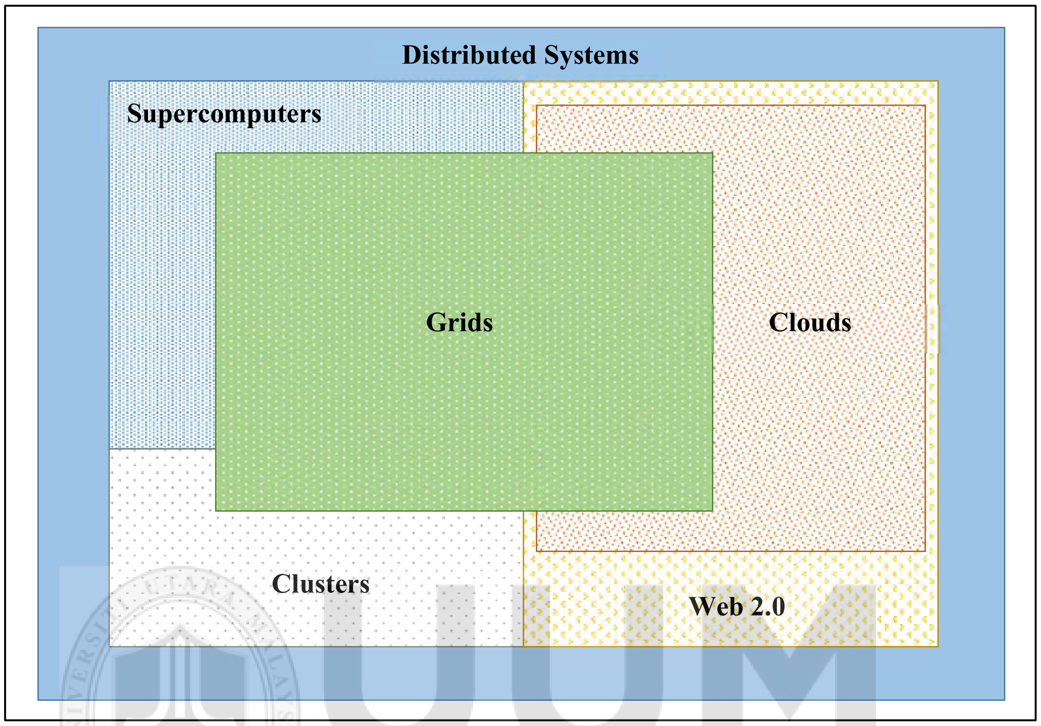


Figure 2.3. Grids and clouds overview

Grid is the backbone of cloud when it emerged from the grid in the modern context in the early 2000's. Cloud can support a grid environment in addition to non-grid environments such as Web 2.0. In other words, grid system can run on cloud system, but cloud system cannot run on grid system (Obali & Topcu, 2015). Detailed comparison between grid computing and cloud computing is listed in Table 2.1. It can be seen that grid computing is designed to provide high performance computing capability with high throughput and low latency which are difficult to achieve in cloud due to shared resources by multiple users demanding different services and reliance on Internet speed and latency.

Table 2.1

High level comparison between grid computing and cloud computing

	Grid Computing	**Cloud Computing**
Business model	Quota basis	Consumption basis
Architecture	Resource sharing	Virtualization
Resource management	Batch-scheduled compute model	Shared by all users at the same time
Programming model	Parallel programming models	Web services APIs
Application model	Supports many applications ranging from high performance computing to high throughput computing	Unable to effectively support applications that require fast and low latency network interconnects
Security model	Decentralized and each grid site has its own administration domain and operation autonomy	Centralized and managed by the same organization

Both grid and cloud offer resource sharing in which cloud uses virtualization infrastructure such as virtual services and virtual machines, and grid uses allocation of large cluster of resources which may be located in the same or different geographical location. In terms of the cost of resource usage, grid uses quota mechanism in which the users or community have certain amount of service units they can spend within a certain time periods while cloud offers flexible costing mechanism which means that the users pay for the services they subscribe (Alkhanak et al., 2016). There are many challenges in cloud computing such as scheduling, load balancing, resource management, quality of service and workload management (Kumar & Kumar, 2019). On the other hand, grid computing has challenges such as job arrival rate, resource utilization, job migration, communication cost and fault tolerance (Khan, Nazir, Khan, Shamshirband & Chronopoulos, 2017).

Krasovec and Filipcic (2019) proposed a model that enhanced grid computing by integrating with public cloud to overcome the complexity and scalability of grid computing system. The proposed model focuses on leveraging cloud infrastructure as a service to support grid computing architecture in improving resource utilization whereby the jobs can be submitted to physical grid resources or to virtual resources in cloud server. This architecture reduces the full dependency on grid resources by allocating some jobs to virtual resources to achieve the same objective with possibly lesser computing power. In this model, the architecture of grid computing is still being preserved while extending its capability to allow user customization, better resource provisioning and scalability. Despite the popularity of cloud computing, grid computing has become significantly important in scientific research projects that require high performance computing capability to cope with increasing demand of computational power (Merelli, 2019).

2.1.2 Job Scheduling and Load Balancing in Grid Computing

Scheduling is one of the main components in the grid service that must be optimized in order to maximize the throughput, minimize processing time and balance the workload of the entire resources. In a grid computing system, resources are distributed throughout a large scale area and each resource has dynamic characteristics and computing capabilities to process submitted jobs (Feng, Weiwei & Xiaomin, 2018).

On the other hand, each job has different characteristics that need to be considered such as job length, size of input and output, deadline and priorities. Due to all these considerations, a good scheduling algorithm should strongly consider the dynamic characteristics of jobs and resources in order to produce the most optimized scheduling process.

There are many algorithms proposed to effectively solve grid scheduling problems. The Priority-based Task Scheduling Algorithm (P-TSA) was proposed by Sun, Zhu, Su, Jiao and Li (2010) to solve scheduling and load balancing problems in grid computing systems with the main focus on minimizing the makespan of processing jobs and maximizing the utilization of resources in grid computing. In the proposed algorithm, all jobs are sorted by the priority values where jobs are classified as a predecessor job (parent job) and successor job (child job). In this case, the successor job cannot be scheduled until the predecessor job is completed. Subsequently, jobs will be sorted according to their estimated completion time (ECT) where the lowest ECT has the highest priority. The sorted tasks then travel through resource machines to record the completion time of each resource machine; the resource machine with the lowest completion time will be selected for scheduling. Experimental results showed that P-TSA has a better performance on the aspect of makespan and resource utilization compared to the other two algorithms by using a random DAG and DAG of molecular code. However, the proposed algorithm only considered the estimated accomplishment time of jobs and estimated completion time of resources but not the characteristics of jobs and resource machines such as bandwidth, job size and historical record of processing state.

The study by Kong et al. (2010) proposed a dynamic grid scheduling algorithm on self adaptive TS to solve scheduling problems in grid computing by reducing the makespan of the processing jobs. The scheduling process in the proposed algorithm was divided into partial scheduling and batch scheduling where information on every partial scheduling based on the Min-Min algorithm is stored in a Tabu list to be used as a simple searching method for these solutions in future search processes. The length of the Tabu list must be considered because if it is too large, it will consume a lot of storage space and require high computation power. Thus, dynamic Tabu length adjustment is implemented in order to produce the ideal length based on the ratio of scheduling time and batch size. In contrast, the scheduling results are not ideal if the Tabu list is too short. The performance of the proposed algorithm was compared to the Min-Min algorithm, Max-Min algorithm and Sufferage algorithm (Siegel & Ali, 2000; Braun et al., 2001). Experimental results showed that the proposed Tabu search algorithm performed better in terms of makespan as compared to the other algorithms. However, scheduling alone is not sufficient in order to provide reliable distributed processing. Thus, it is essential to also consider load balancing and fault tolerance aspects.

Hybridization between Cuckoo Search (CS) and Genetic Algorithm (GA) for job scheduling in grid computing was introduced by Sajedi and Rabiee (2014) to minimize the completion time and prevent trap in a local minimum. It is claimed that GA suffers from long processing time to perform required test to obtain optimal parameters, and this disadvantage is covered by the advantages of CS which has faster convergence and has the ability to avoid local minimum. The first stage involves generating a population of Cuckoo eggs that have greatest change for further growth. This population will undergo mutation and crossover during the second stage to produce new population which is also optimal solution. The experimental results showed that the proposed hybrid algorithm achieved the lowest completion time when compared with CS and GA. It was assumed that all machines are always available and jobs do not have time constraint to be completely processed, and these facts could become factors that the proposed algorithm may not work efficiently when interruption happens.

Adaptive workflow scheduling that involves initial static scheduling, resource monitoring and rescheduling was proposed by Garg and Singh (2015) with objective to minimize execution time. Before the scheduling begins, set of available resources and loads are identified, and this process is continuously executed to update the list of available resources. Then, DAG will be used to perform initial static scheduling to map the workflow task to suitable resources and the scheduling results will be submitted to the execution manager for further submission to the resources. On the other hand, the resource monitor periodically checks for abnormalities such as load increment or new resources available. These events will be exchanged with workflow task scheduler which will then decide whether to reschedule the workflow task to other resources or do nothing. Despite the performance of the proposed algorithm in terms of minimizing the execution time, it did not consider the dynamic resource availability on the executing task which could lead to the inability to perform rescheduling when the resource executing current task unexpected becomes unavailable.

Sheikh, Shahid and Nagaraju (2017) proposed a dynamic task scheduling strategy through enhancement of task level parallelism to minimize the makespan. The first phase of the proposed algorithm is to divide task into multiple subtasks within the maximum task limit. In the second phase, each subtask which is also treated as individual task, will be submitted to available resources dynamically in parallel manner until all the subtasks for a specific task are completely executed. The next task submitted by the user must wait for the current task to be completely processed before it can undergo the splitting and submitting phases. The results showed that the proposed algorithm achieved significant reduction of makespan. It is clear that parallelism can significantly reduce the processing time of a large task but the condition where the next task must wait for the current task to be completely executed is quite risky especially when some subtasks of the current task failed to complete and eventually increase the latency of the next tasks to be executed.

Younis and Yang (2018) proposed two hybrid metaheuristic scheduling algorithms in reducing the makespan of the scheduling process in grid computing. The makespan value is used as a benchmark where the small makespan value indicates the high utilization of the available resources. In this research work, the traditional Variable Neighborhood Search (VNS) has been improved by introducing four new structures which are penalty-based move, penalty-based swap, longest max to min move, and random max to min move, based on the concepts of transfer and move of assigned jobs to or from the selected resources. The first proposed scheduling algorithm combines ACO and VNS while the other algorithm combines GA and VNS. The proposed ACO-VNS algorithm finds the best resources by using the free earlier value when calculating the heuristic function along with the pheromone value. Then, the solution found by the local best ant will be improved by using the improved VNS structures. The second proposed scheduling algorithm combines GA and VNS where the improved VNS structures deployed as a mutation operator in the GA algorithm encourage the exploration of the search space in minimizing the makespan. Experimental results showed that GA-VNS has the lowest makespan followed by ACO-VNS when compared to the other scheduling algorithm. However, neither algorithm considered the failure problem and only evaluated in terms of makespan instead of other performance metrics such as throughput and success rate.

One of the most challenging problems in resource management is load balancing. Load balancing is important to consider since it ensures fair job distribution across all available resources (Rathore & Chana, 2014; Khan, Nazir, Khan, Shamshirband & Chronopoulos, 2017). It ensures all submitted jobs are distributed equally to each resource, minimizes the execution time of each job and maximizes resource utilization. To be specific, this is done by minimizing the difference between heaviest loaded node and lightest node. As a result, stagnation problems can be effectively resolved as the probability of a resource being overloaded with jobs while the others remain idle is minimized through the load balancing policy. Furthermore, the load balancing should be self-adaptive to automatically adjust with dynamism and heterogeneity of the resources (Darmawan & Aradea, 2018).

There are many proposed algorithms that take into consideration load balancing along with scheduling problems in grid computing. For instance, the heuristic-based load balancing technique by using ACO was proposed by Sharma, Sharma and Dalal (2014). By using the ACO algorithm, a job will be assigned to the resource that has the highest transition probability during scheduling. After the execution is done, regardless of whether it is a success or failure, the pheromone intensity of the current resource will be updated. Throughout this approach, a resource that recently completed its processing will have less pheromone which will directly control the value of transition probability during the scheduling process. In the end, other resources will also have the possibility to be chosen to process the remaining list of jobs instead of simply one or a few powerful resources. The proposed algorithm was evaluated against the random resource selection algorithm and the results achieved lower execution cost and time. However, despite considering the load balancing aspect, it was not measured thoroughly in the experimental results.

Batch mode scheduling strategy was proposed by Maipan-uku, Konjaang and Baba (2016) with the aim to reduce makespan, increase resource utilization and balance the load. Before the scheduling process begins, all tasks are sorted in ascending order which means that task with lowest expected completion time will be placed in front of the queue. Then, the average completion time will be collected from all available tasks and whichever tasks that have expected completion time greater than average completion time will be scheduled first, followed by the tasks that have expected completion time equal or lower than average completion time. The proposed algorithm was compared with Min-Min algorithm that considers tasks with minimum expected completion time which assign the tasks to resources that yield minimum completion time. Results showed that it achieved lower makespan, higher average resource utilization and load balancing. In spite of the good performance, the algorithm works well when failure is not part of the consideration but when it is being considered, the expected completion time alone is not enough to ensure effective scheduling in faulty system.

Khan (2017) proposed an effective load balancing and dynamic group scheduling that used resources partitioning to reduce job processing time and increase resource utilization. All resources are group into several partitions in which each partition has one super node and several resources. Super node of each partition controls the job pool and communicate with other super nodes from different partition to exchange information while resources within the partition can only communicate with each other and the super node. In terms of dynamic jobs allocation, linear programming model is used to estimate the execution time of each combination of jobs and resources before the job can be submitted to the respective resource within a specific partition. The proposed algorithm was compared with first come first serve and ACO algorithms and results showed that it achieved lower processing time and higher resource utilization. Notwithstanding the results, it was not specified whether the resources that reside within a partition can be re-assigned to another partition which this could lead to imbalance fitness of partitions when one partition might have, and whether super node can be changed from time to time depending on its load.

Dynamic load balancing with advanced reservation was proposed by Sheikh, Nagaraju and Shahid (2018) to minimize task waiting time and load imbalance by considering the resource load prior to allocating the task. The advanced reservation is performed by first checking the resource availability and running time to execute the tasks. At certain time intervals, running time will be added into the load share queue as a reference to decide which resource is overloaded or underloaded. On the other hand, whenever the task is executed partially or completely, the amount of executed length will be deducted from the load share queue which will indicate that the resource is ready to receive more tasks. The proposed algorithm resulted in higher load balancing and lower makespan and claimed to be suitable for applications that require minimal task waiting time but not turnaround time.

Abdullah, Ali and Haikal (2019) proposed TOPSIS-based multi-criteria and hierarchical load balancing to effectively improve the load balancing in computational grid environments. The n-level architecture model has leaf level, intermediate level and root level which are interconnected hierarchically, and static or dynamic information of all resources within this model are stored in a global information system. The jobs are assumed to be generated from one local resource and will be prioritized by the local scheduler for local processing. If the job cannot be completed locally within a time constraint, it will be routed to the global scheduler which distributes the job to other resources that can execute within a time constraint inside the global cluster. In terms of fault tolerance capability, it was noted that the non urgent job will be routed to faulty resources and is expected to fail to undergo the rescheduling process to a better resource. It was also highlighted that the proposed algorithm should be coupled with fault tolerance techniques such as a checkpoint-based resubmission process to be more effective. Their proposed algorithm resulted in lower average completion time, higher throughput and maximum load balancing when compared with minimum completion time (MCT) and user demand aware grid scheduling models.

2.1.3 Issues and Limitations of the Scheduling and Load Balancing in Grid Computing

Scheduling is definitely one of the main components in the grid environment. Effective scheduling ensures that each job is assigned to the best resource. Most related works often consider job processing time. However, in order to provide reliable and accurate scheduling, it is important to consider other job and resource characteristics such as size, current load, current bandwidth, availability and so on.

Furthermore, the scheduling process simply assigns jobs to suitable resources without balancing the load. If, out of all available resources, there is only one best resource detected by the scheduling policy, the same resource will be assigned with jobs continuously while the other resources will remain idle (Rathore & Chana, 2014). Therefore, it is definitely crucial to also incorporate load balancing with job scheduling in order to avoid stagnation and provide fair resource utilization. Load balancing ensures that if the load of a resource is not within a normal state, the resource will have low probability to be chosen during the scheduling process.

Another issue that should be considered during the scheduling process is resource availability. It is possible that the resource information in the grid information service (GIS) is not up-to-date and may lead to the scheduler performing a bad scheduling decision when the resource is temporarily unavailable. Increasing the frequency of information update in GIS could be the solution but this will increase system overheads. In addition, checking the resource availability before submitting the job is another good approach to reduce the possibility of unavailable resources being assigned to jobs (Sheikh, Nagaraju & Shahid, 2018).

Table 2.2 summarizes the work related to job scheduling and load balancing in grid computing in terms of the main objectives and open issues. It can be seen that minimizing makespan is the most common objective as it indicates how quick an individual task can be executed completely. The least common objective is load balancing because in a system without faults, the scheduling process that would indirectly consider load balancing can be performed easily due to the fact that each individual resource has more predicted capabilities and fitness. However, in a system with faults, resource capabilities and fitness are unpredictable and may change from time to time. Majority of fault tolerance algorithms focus on reducing faults and often, this requires jobs to be submitted to fit resources and eventually leads to unfair jobs distribution and poor resource utilization.

Table 2.2

Summary of literature related to job scheduling and load balancing in grid

Authors	**Objective**	**Drawback**
Kong et al. (2010)	Reduce the makespan of the processing jobs	Does not consider load balancing
Sun et al. (2010)	Minimize the makespan of processing jobs and maximize the utilization of resources	Does not consider bandwidth, job size and historical record of processing state
Sajedi & Rabiee (2014)	Minimize completion time and avoid local minimum	Unable to work efficiently when interruption happens
Garg & Singh (2015)	Minimize execution time	Did not consider dynamic changes to the resource availability on executing task
Sheikh, Shahid & Nagaraju (2017)	Reduce makespan	Dependency of next task to the previous task execution may lead to latency
Younis & Yang (2018)	Reduce the makespan of the scheduling process	Does not consider throughput and load balancing

Sharma, Sharma & Dalal (2014)	Reduce the average execution time and cost of the tasks	Considers load balancing but not validated in the experimental results
Maipan-uku, Konjaang & Baba (2016)	Reduce makespan, increase resource utilization and load balancing	Reliance on expected completion time along is not sufficient in faulty system
Khan (2017)	Reduce the processing time and increases resource utilization	Resource within partition cannot be moved to other partition and lack of dynamic super node nomination
Sheikh, Nagaraju & Shahid (2018)	Load measurement before task execution to efficiently distribute load prior to task execution	Suitable for system that needs minimum waiting time but not turnaround time
Abdullah, Ali & Haikal (2019)	Load measurement before task execution to efficiently distribute load prior to task execution	Does not consider failures

2.1.4 Grid Computing Simulation Tools

Grid computing simulations tools are developed to overcome the challenges to deploy real grid environment which is costly and involves complicated infrastructure setup. In addition to that, the real infrastructure has limitations such as low scalability and reconfiguration possibility, and is inflexible to support hardware components and topology changes. There are several grid simulations tools available such as MicroGrid, Bricks, SimGrid, GangSim and GridSim (Mollamotalebi, Maghami & Ismail, 2013). In addition to that, there are several popular grid simulation tools based on the number citations in descending order which are GridSim, SimGrid, OptorSim and GangSim (Prajapati & Shah, 2015).

MicroGrid was introduced by Song et al. (2000) to allow establishment and evaluation of grid computing middleware, applications and services. It allows repetitive and controlled scientific experiments which run on virtual resources that are heterogenous physically. The MicroGrid is modeled to support applications developed using Globus toolkit, but it requires significantly huge amount of time to execution experiments as applications run on emulated resources (Buyya & Murshed, 2002). MicroGrid was used by Xia, Dail, Casanova and Chien (2004), Liu, Xia and Chien (2004), and Xhafa, Carretero, Barolli and Durresi (2007).

Bricks was developed by Takefusa, Matsuoka and Nakada (1999) to support performance evaluation of scheduling schemes in computing environments such as grid and cluster. It allows simulation of computing systems behaviors that include scheduling, network topologies and processing plans. Its architecture is designed in such a way that components are replaceable when needed to accommodate various evaluations. Generally, it consists of two main units which are grid computing environment and scheduling unit. Client, network and server are part of grid computing environment, while applications and services such as database, monitoring, predictor and scheduler are part of scheduling unit. Bricks was used by Takefusa, Casanova, Matsuoka and Berman (2001).

Gangsim was an enhancement of Ganglia monitoring toolkit that incorporates instances of virtual organization with simulated components developed by Dumitrescu and Foster (2005). It focuses on simulating policy driven management infrastructure such as the number of CPUs and their time, network bandwidth and disk space. It supports capturing realistic grids behavior at high level, and not detailed behaviors of scheduler and jobs. Due to its immaturity, it was not implemented in the actual research experiments.

SimGrid toolkit was firstly introduced by Casanova (2001) to allow simulation of computing application scheduling in heterogenous and dynamic grid environment more realistically. It was also proven to generate a more correct and accurate simulation results. Computing resources are treated as independent resources without a need for interconnection topology which allows simulation of wide range of computing environments as the users have flexibility to specify their topology requirements. According to Prajapati and Shah (2015), SimGrid was the second most popular grid simulation tool based on the number of citations by researchers. SimGrid was implemented by Lebre, Legrand, Suter and Veyre (2015), Hirofuchi, Lebre and Pouilloux (2015), Brennand, Duarte and Silva (2016), and Fanfakhri, Yousif and Alwan (2017).

GridSim is an open source platform developed by Buyya and Murshed (2002) which has almost the same features as SimGrid in terms of ability to model heterogenous resources in addition to extensible information system that can store and query properties of the resources for designing resource discovery system. It allows simultaneous tasks execution to the same resource and supports both static and dynamic schedulers. The GridSim has a layered architecture where each layer has specific functions. One notable component of GridSim is the grid resource broker which is responsible to receive submitted tasks and apply scheduling policies. Each user is connected to an instance broker, and all submitted tasks will go through the broker instead of direct access to the resources. Based on analysis performed by Prajapati and Shah (2015), GridSim is the most popular grid simulation tool due to highest citation counts and implementation in recent works such as Idris et al. (2017), Shukla, Kumar and Singh (2018), Garba et al. (2020), and Eng et al. (2020).

Based on the list of grid simulation tool, GridSim is widely used by researchers in simulating their works due to its architecture that is close to the actual grid environment and flexibility to support wide range of simulation experiments. In addition to that, it is not only used for simulating grid environment, but also other application domains such as high performance computing (Eleliemy, Mohammed & Ciorba, 2016), cluster computing (Gabaldon, Guirado, Lerida, & Planes, 2016) and autonomous driving simulator (Trasnea et al., 2019).

2.2 Fault Tolerance

Fault tolerance is a method to keep the system working optimally even if any of its components are in a faulty status. A good fault tolerance system must deal with the availability, safety, reliability, and maintainability factors (Smith, 2017). The system must be available and ready to serve the user in the given time and at the same time can be reliable to work constantly over a long period of time with minimal disruption. A good fault tolerance system also has a high maintainability system and can deal with system failure without affecting the quality of the outputs.

2.2.1 Fault Tolerance in Grid Computing

Fault tolerance is the ability of a system to perform its function correctly even in the presence of failure (Garg & Singh, 2011). Fault tolerance management is the process to identify and handle failures in grid computing (Farid & Hussain, 2017). This process includes identifying available failures and supporting reliable execution in the presence of failures (Keerthika & Kasthuri, 2012). In grid computing systems, there are dynamically changing conditions where resource performance changes from time to time, a resource may become unavailable without any notification and network connections become unreliable. Thus, it is important to define proper fault tolerance strategies and techniques to be used in designing the most reliable fault tolerance algorithm.

Fault tolerance is one of the important issues highlighted in the distributed system. Distributed systems such as grid computing and cloud computing use multiple resources that are connected by a network to provide a high performance computing capability that cannot be achieved by a single computer. One of the main concerns in ensuring the performance of a distributed system is the way it handles the failure of one or multiple resources in real time. An efficient fault tolerance system can detect the faults and has the ability to recover from them without causing fatal failure to the system that requires user intervention.

In grid computing specifically, fault tolerance system must be able to adapt with dynamic changes of the resources and executing jobs so that appropriate actions can be taken to ensure all the jobs can be completely reprocessed despite the presence of faults (Alzboon, Arif, & Mahmuddin, 2016). In addition to that, a good fault tolerance system must have the ability to apply heuristic learning to improve the scheduling decision and jobs reassignment. As illustrated in Figure 2.4, the fault tolerance system in grid computing consists of five basic components which are shared grid host, scheduler, GIS, fault handler and grid resource broker. Shared grid host provides user interface for the grid user to submit jobs and retrieve the results. The scheduler is responsible to perform allocation decision based on user requirements. On the other hand, GIS contains information of all available resources such as processing element (PE) rating, number of PE per machine, and number of machines per resource.

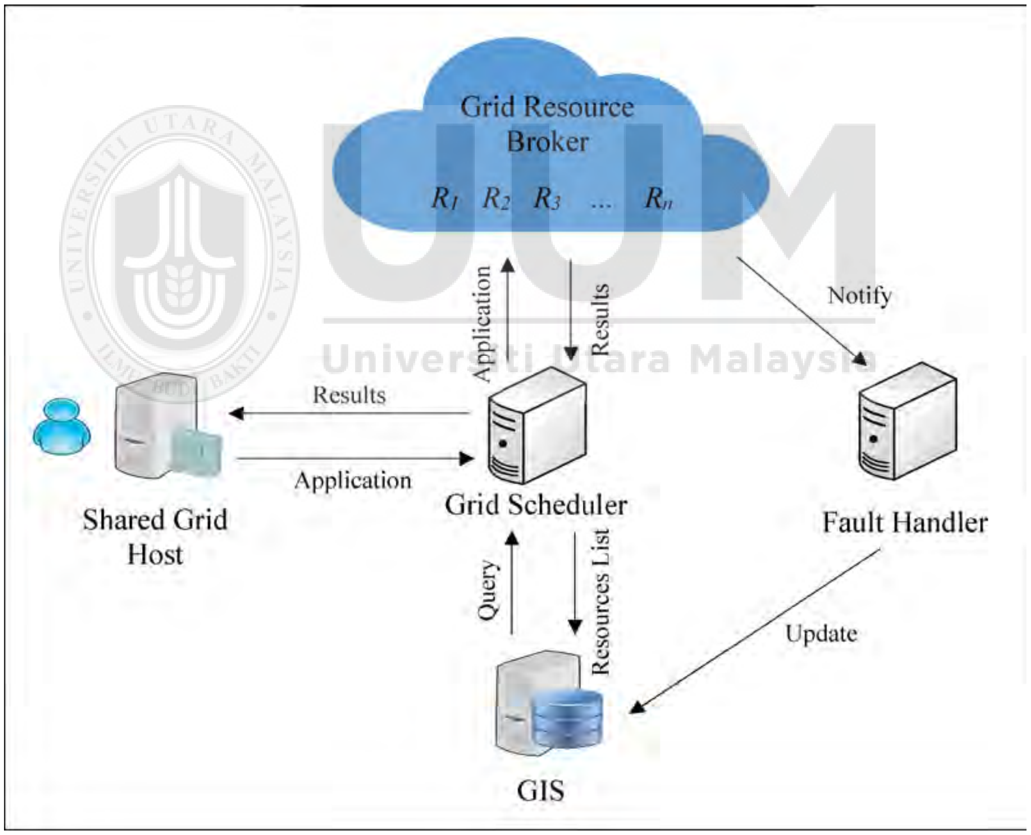


Figure 2.4. Basic architecture of fault tolerance system in grid computing

The main component of fault tolerance is the fault handler which is responsible for detecting and mitigating failures by initiating fault tolerance techniques. Last but not the least is the grid resource broker that groups and manages all the resources which may or may not reside in the same physical locations to take up submitted jobs. Section 2.2.1 discusses in detail the existing fault tolerance strategies applied in grid computing systems.

One of the most common fault tolerance strategies is fault detection which leverages fail signal or acceptance test in order to detect failures (Balasangameshwara & Raju, 2012). Once detected, a proper mitigation plan will be executed as defined by the user. In dynamic environments, early error detection is very suitable to implement as error detection is done prior to submitting jobs to resources. In case any abnormalities are detected in a chosen resource, a migration plan will be executed by assigning another suitable resource to process a particular job (Rakheja, Kaur & Rkheja, 2014). This process will be repeated in each cycle, eventually reducing the possibility of error by assigning the job to the most reliable resource. Early detection is also related to fault prevention in which it is used to prevent faults and avoid the possibility of malfunctioning of resources. This process is often being applied during the scheduling and execution process. However, fault prevention techniques should be applied carefully to avoid overheads caused by excessive preventive measures which will eventually lead to inefficiency of the grid environment (Balpande & Shrawankar, 2014).

In addition to detection and prevention, a recovery strategy must be applied because all jobs are meant to be processed completely. Recovery refers to the process of recovering failed jobs so that they are completely processed in the end and also recovering failed resources so that they will be back online and fully functioning. Retry and alternate resource techniques are commonly applied in recovery mode due to their simplicity. Qureshi et al. (2011) proposed a hybrid fault tolerance technique that is based on a combination of a simple alternate task with retry technique and task level checkpoint technique in which it inherits the best characteristics of both techniques. When the task fails for the first time, alternate task with retry technique will be invoked which means that the same task will be resubmitted using different execution characteristics. If the alternate task fails, the checkpoint manager would record the checkpoint before resubmitting the failed alternate task to the same resource. If the resubmitted alternate task fails for a second time, the system will resubmit the remaining incomplete task from the last checkpoint to another resource. The comparison was done between alternate task with retry and alternate task with checkpoint and the results showed that the checkpoint technique increases throughput and reduces turnaround time significantly. Regardless of the positive results from the experiment, the proposed algorithm does not seem to consider balancing the load of the entire system and applying the resource suspension technique which is important in order to avoid resources with a bad historical record to be assigned in the next execution.

Fault tolerance time to release scheduling algorithm, which is based on transmission time and fault rate, was proposed by Keerthika and Kasthuri (2011). This algorithm considers user deadline and executes the job within an expected deadline by assigning it to the most suitable resource. The time to release (TTR) is calculated for each job and resource combination. By comparing the value of TTR against the expected deadline, the job will only be submitted to the resource that has a TTR lower than the expected deadline. This ensures that the resource is capable of processing the job which eventually increases the hit rate or successful rate. However, this algorithm did not address the action needed to be taken during job processing failure which is very important because every job is meant to be processed successfully in the end. Additionally, it is also possible for the pool of resources to have a TTR of more than the expected deadline of all jobs due to limited bandwidth, limited processing machine and so on. Thus, it should be further extended to support dynamic scheduling which is aligned to the nature of the grid environment.

The Fault Tolerance Min-Min policy, proposed by Keerthika and Kasthuri (2012), is proven to perform better with less makespan in the presence of failure and increases the number of successfully completed tasks. Makespan is the total time taken to completely process a set of jobs. Based on this algorithm, Keerthika and Kasthuri (2013) further improved its functionality by proposing the Bicriteria Scheduling Algorithm (BSA) which considers user satisfaction with a proactive fault tolerance method to reduce makespan, achieve better hit rate and higher user satisfaction. The BSA algorithm first constructs several matrices of jobs and resources such as expected time to compute, communication time and total completion time. Then, the failure rate of each resource will be calculated by dividing the total number of failed tasks with the total number of submitted tasks. The failure rate will be used to determine the best resource to execute the next task in the queue. Finally, the update will be carried out on the matrix by removing the completed task from the list. The results showed that the proposed BSA achieved a better hit rate and makespan than the Fault Tolerance Min-Min algorithm. Even though the proposed algorithm is effective in increasing the hit rate, it does not seem to have control over load balancing which means that resources with the lowest failure rate will potentially be overloaded. Furthermore, the proposed algorithm can be further extended by considering resource load and resource availability to improve its efficiency in scheduling and fault tolerance.

Application checkpointing with replication in the grid was proposed by Bawa and Singh (2012) to tackle the problem where a failed job needs to be re-executed from the beginning in the presence of failure which would increase execution time significantly. The proposed algorithm consists of managers and executers. The manager acts as a central point which keeps track of available executers through heart beat signal as well as storing checkpoint data. In addition, the replication manager is also present which will replicate checkpoint data from the initial manager. In contrast, executers will first determine whether checkpoint data exist in the database after it receives the job or thread from the manager. If data exist, checkpoint data will be restored, and execution will start from a previously saved checkpoint, or else execution will start from the beginning. In case of any failure that is causing checkpoint data to become corrupted, it can also be restored from the replication manager. Even though the proposed algorithm works effectively in a faulty environment, it creates a checkpoint overhead in the fault free environment as well as undertaking small tasks execution which means that the execution time may increase as the initial execution has to undergo all pre-execution processes before being executed.

Balasangameshwara and Raju (2012) proposed a fault tolerance load balancing algorithm to minimize response time and optimize node utilization. The submitted job will go through the local grid scheduler which replicates the job and discovers potential resources before sending it to the load balancing decision maker. The decision maker will decide whether to process the job locally or remotely. Regardless of local or remote execution, the job will be forwarded to the fault detector which evaluates the availability of any proposed resource and if the proposed resource is not available, a failure message will be triggered to the fault manager which will activate the replicated job in the grid scheduler to undergo the rescheduling process. The grid scheduler uses a threshold which is calculated based on the load and demand for resources to decide which resources to assign to process the job. The drawback of this proposed algorithm is that the job will be rescheduled from its initial state which will lead to longer execution time if it fails continuously.

The fault tolerance checkpointing system, proposed by Amoon (2013), considers the resource failure rate and average failure time to define the checkpoint interval for each job. In this system, resources will first be sorted based on response time, failure rate, and average failure time and then job assignment will be performed based on the sorted list. In the proposed algorithm, a failed job will be restarted from the last saved state which avoids time waste in reprocessing the failed job from the beginning. This process is handled by the checkpoint server that stores the snapshot of a partially completed job and checkpoint handler that is responsible to dispatch the job and retrieve checkpoints from the checkpoint server. Results showed that the proposed algorithm achieves higher throughput, lower turnaround time and lower failure tendency with a significantly low number of checkpoints. Instead of considering the load balancing, the scheduling process in the proposed algorithm is biased towards fit resources which would lead to unfit resources and never getting the job assigned.

Rathore (2015) proposed priority-based scheduling with load balancing using fuzzy rules to increase resource utilization and decrease task execution time. Each task will be assigned a priority based on length and input file size. The task priority can be static which will be kept for the entire life, or dynamic which may be changed by the scheduler from time to time. In addition to defining the task priority, resources are grouped into three groups whereby each group is set to be assigned with a specific task priority. During execution, the grid scheduler will check the availability of resources before assigning a task to ensure no case will arise where an unavailable resource is assigned to a task. However, the scenario where the resource fails to complete the task processing but still shows as an available status is not considered. This may lead to unfit resources being assigned to tasks even though the resource's availability is not equivalent to its fitness which will eventually lead to an increase in execution failure despite having good load balancing.

Rathore and Chana (2015) proposed a threshold-based hierarchical load balancing technique in grids by categorizing the load into several categories such as underload, overload, light load and normal load. Each category has its threshold value defined by the load deviation dynamically. This is referenced by the grid scheduler to distribute the job according to the load and perform job migration from an overloaded node to a lightly loaded node and underloaded node using the random selection strategy. The selection of resource is based on whichever resource has the minimum load. In terms of failure handling, a backup will be created by the local grid scheduler for each submitted job which is saved in a remote grid scheduler before node searching is initiated. Whenever failure happens, the fault manager will activate the backup by sending a signal to the remote grid scheduler to re-queue the backup into the local grid scheduler. The proposed algorithm results in lower response time, higher resource allocation efficiency, and lower communication overhead and makespan. Despite good results, the fault tolerance approach was not validated in the experiments and the backup activation approach requires more time as the failed job will be reprocessed from the initial state.

Singh and Bawa (2016) proposed the proactive fault tolerance algorithm for job scheduling that proactively focuses on preventing failure rather than curing it. Each resource will have its own performance index that is calculated based on historical information, workload, availability, response time, mean time to failure and mean time between failures. The performance index indicates the fitness of a resource and will be used as a reference during the scheduling process. In terms of fault tolerance capability, checkpoints will be captured on every 30% completion of every task. During the execution, the monitoring agent will continuously validate the performance index of executing resources against the threshold and, if it exceeds the threshold, the task may be shifted to other resources based on calculated shifting costs. The proposed algorithm was validated against a post-active heuristics algorithm and the results showed that it produced slightly lower execution time, lower faults rate and lower execution cost. Despite the promising results, the proposed algorithm does not employ the reactive action to reschedule tasks that failed during execution.

Failure aware scheduling algorithm based on incremental checkpoint scheme was proposed by Singh (2016) to overcome the checkpoint overhead without affecting the system performance. The scheduling process considers the resource performance as well as failure rate to generate its capacity. Based on the capacity which may change from time to time, the tasks that are sorted in decreasing order based on load will be assigned accordingly. In terms of fault tolerance, full checkpoints are coupled with incremental checkpoints that only capture the change from the last incremental checkpoint. Using this method, checkpoint overhead can be reduced as the reprocessing of failed task will happen from the last incremental checkpoint instead of from the last full checkpoint. On the other hand, by reducing the number of full checkpoints, the system memory can be reallocated for other purposes. The results showed that the proposed algorithm achieved better average response time when compared with speed only scheduling algorithm. Nevertheless, the experiments were conducted using small number of tasks and it is not entirely proven that the proposed algorithm is effective in processing large number of tasks.

Haider and Nazir (2017) proposed a hybrid fault tolerance scheme based on proactive and reactive approaches as employed in existing fault tolerance mechanisms. The resource selection process is performed using a proactive technique while a reactive technique will be initiated for handling faults. The proactive technique starts with resource filtration based on location, availability and reliability and is followed by identification of optimal resources using GA based optimal resource identification. On the other hand, the reactive technique consists of failure prediction that detects possible failure based on hardware temperature and failure detection that receives information from the failure prediction component and from the hardware. The standard checkpoint interval is set to every 25% of job completion and will be reduced accordingly (every 5% of job completion) based on information from the failure prediction component. Failed jobs will be resubmitted from the last checkpoint maintained by the checkpoint manager with minimized recovery time. The results showed that the proposed algorithm has higher throughput, lower average waiting time, average turnaround time and better efficiency (cost, energy and time). Despite the promising performance, the implementation of direct resource filtration could cause resources that do not meet the criteria to never be assigned to jobs and, eventually, lead to poor load balancing.

Hierarchical organization model for computational grid that offers grid scheduling, load balancing and fault tolerance was proposed by Abdullah, Ali and Haikal (2017) to introduce self repairing n-try dynamic hierarchical grid model for scheduling and replication of master resource on its child for load balancing. All the resources are organized in hierarchical form with multiple levels such as root level, intermediate level that consists of clusters, and leaf level that consists of community or child of individual cluster. Jobs submitted by the users locally will be executed by their own resource but it that cannot be completed within the time limit, the job will be submitted to the global scheduler which will assign the job to suitable resource within the cluster or within the community. In terms of fault tolerance, any failed resource will be replaced by its replica which is any of its child resource so that the hierarchical structure of all resource can be maintained. Results showed that the proposed algorithm achieved lowest average completion time and communication overhead, and highest tree stability ratio. Notwithstanding the performance of the proposed algorithm, the focus is to overcome resource failure rather than job failure.

Fault tolerance nearest deadline first scheduled was proposed by Goswami and Das (2018) to provide periodical runtime backup to support job reprocessing from the last backup point to other alternative resources. This approach has the same concept as resubmission using checkpoint, which is commonly used in fault tolerance algorithms. Resources are grouped into the categories of underutilized, less loaded and overloaded which are used by the broker to define resource ranking. Each individual job carries parameters that are referenced by the broker to find matching or higher capacity resources to process the job even during the reprocessing phase. The proposed algorithm is claimed to save resubmission time as well as execution time. However, the experiments were done using three clients, two resources and one broker which is considered very small scale and insufficient to conclude its performance for large scale infrastructures.

The resubmission-based fault tolerance approach for jobs scheduling was proposed by Ahuja and Banga (2019) that uses a replica-based approach to overcome the limitation of the checkpoint approach that incurs overhead and time wastage. Before getting assigned to respective resources, a task will be divided into several subtasks by the subtask generator and managed by the subtask manager within GIS. Each subtask will be processed independently and in parallel. During failure, each subtask will be resubmitted to a different resource from the initial state instead of using the checkpoint approach. It is claimed that this approach saves a lot of time and reduces overhead which will eventually lead to better performance. Despite the promising approach, the experiments were done using s small number of tasks and resources and did not consider the load balancing of the system which is crucial in preventing stagnation.

2.2.2 Issues and Limitations of Fault Tolerance in Grid Computing

Faults will lead to errors that cause failure. In other words, preventing faults may reduce system failure as system failure is more difficult to deal with without human intervention. There are several techniques in identifying faults which include the push model, pull model, probability-based techniques and neural network-based approaches. Open issues related to fault tolerance include errors, failures and faults detection and handling. These issues can be addressed via several strategies such as the effective fault identification technique, use of fault tolerance scheduling technique to perform resource allocation, logging problems for further analysis and improvement to better predict failures and impact to the system performance, and use of the hybrid fault tolerance technique that combines the best characteristics of multiple algorithms (Haider & Nazir, 2016).

Reducing makespan, execution time, and resubmission time, as well as increasing success rate have been key objectives in fault tolerance. All these can be achieved by having the most effective fault tolerance algorithm, single or hybrid, that can overcome fault in less possible complexity to avoid overhead to the system. At the same time, other internal aspects such as resubmission or re-execution strategies, reliability of the checkpoint manager, optimal resource determination, and reoccurrence of failure prevention should not be neglected. For instance, recently failed resources should not be reassigned with jobs as the same failure could continue to happen because the resource is no longer fit. The checkpoint manager must be reliable in order to ensure that the checkpoint information or stored processed jobs will not be corrupted or may lead to overheads (Ahuja & Banga, 2019). Discrepancies to checkpoint information or saved jobs may lead to bad outputs and could be even more difficult to fix.

Checkpoint technique is effective in reducing the reprocessing time when a job fails to be processed completely. However, having too frequent checkpoint records may lead to runtime overhead and eventually increases the makespan (Garba et al., 2020). On the other hand, not having enough checkpoint records may reduce the benefit of having a checkpoint technique as the amount of time saving is not so significant. Thus, it is important to properly control the amount of checkpoints to obtain full benefits and improve the system performance in the presence of faults.

Another important issue in fault tolerance is load balancing. Typically, load balancing is being disregarded due to the primary focus on applying fault tolerance techniques. Ultimately, the system may be good at handling faults but inefficient in ensuring load balancing in the presence of faults. Thus, resource determination during initialization or resubmission should also consider the fitness or execution history of each resource in addition to the current load in order to tackle both fault handling and load balancing (Idris et al., 2017).

Table 2.3 summarizes the works related to fault tolerance in grid computing in terms of the main objectives and open issues. Most of the fault tolerance algorithms do not consider the load balancing aspect as there is a big trade-off between achieving good load balancing and minimizing makespan. To achieve the minimum makespan, tasks should be distributed to the fit resources and, often, this method would reduce the utilization of resources. On the other hand, to achieve good load balancing, both fit and unfit resources should be assigned with tasks based on their capacities and execution history to increase resource utilization.

Table 2.3

Summary of literatures related to fault tolerance in grid

Authors	**Objective**	**Drawback**
Qureshi et al. (2011)	Increase throughput and reduce turnaround time	Does not consider load balancing and resource execution history

Keerthika & Kasthuri (2011)	Increase hit rate	Does not address the situation where most or all resources have time TTR than expected deadline
Balasangameshwara & Raju (2012)	Minimize response time and optimize node utilization	Failed jobs rescheduled from the initial state
Bawa & Singh (2012)	Reduce execution time	System overhead due to implementation of checkpointing and replication
Amoon (2013)	Increase throughput, lower turnaround time and lower failure tendency with significantly low number of checkpoints	Bias towards fit resource, does not consider load balancing
Keerthika & Kasthuri (2013)	Reduce makespan, better hit rate and user satisfaction	Does not consider load and resource availability
Rathore (2015)	Increase resource utilization and decrease execution time	Does not consider resource fitness
Rathore & Chana (2015)	Reduce communication overhead and makespan, and increase resource allocation efficiency	Job backup at local resource may increase memory overhead
Singh (2016)	Reduce checkpoints overhead	Validated on small number of tasks but not large number of tasks
Singh & Bawa (2016)	Prevent failure rather than cure	Does not employ the reactive action to reschedule tasks that failed

Abdullah, Ali & Haikal (2017)	Overcome resource failure using resource replication technique	Focus on resource failure rather than job failure
Haider & Nazir (2017)	Reduce cost, energy and time	Does not consider load balancing
Goswami & Das (2018)	Save resubmission time and execution time	Applicable only to small scale system
Ahuja & Banga (2019)	Overcome the limitation of checkpoint approach that incurs overhead and time wastage	Does not consider load balancing

2.3 Ant Colony Optimization

Years ago, researchers started to investigate the behavior of real ants such as foraging and nest construction. For instance, a double bridge experiment which was conducted by Goss, Aron, Deneubourg and Pasteels (1989) to investigate the foraging behavior of real ants. Ants move in a continuous path from nest to food source as shown in Figure 2.5 (a). When there is an obstacle, as shown in Figure 2.5 (b), ants will randomly decide to turn right or left without knowing which direction has the shortest path, as shown in Figure 2.5 (c). It is known that whenever an ant traverses, it will deposit a chemical substance called pheromone that evaporates at a certain rate along the way. By assuming that each ant moves at the same speed and takes the same route to return to the nest, the shortest path will have more pheromone proportionally with the number of traversed ants. The level of pheromone on each path will be an attraction factor for incoming ants and the optimal path will be introduced after a certain cycle, as shown in Figure 2.5 (d).

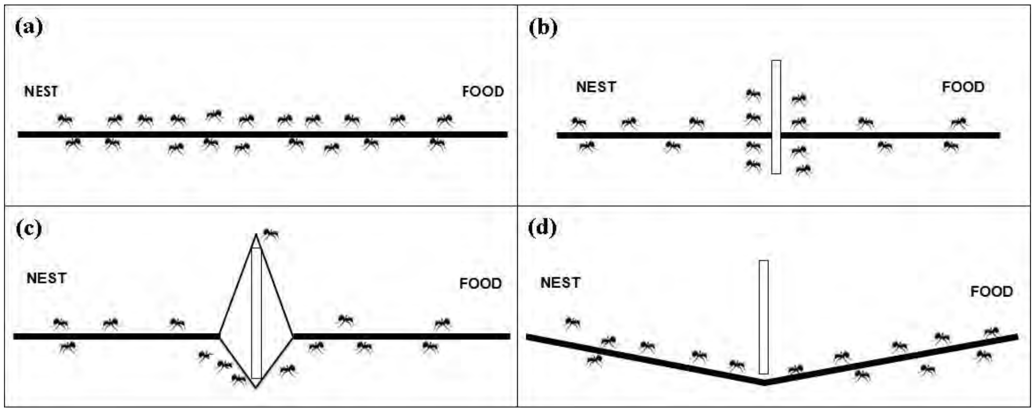


Figure 2.5. Ants behavior in foraging process (Perretto & Lopez, 2005)

ACO is a biologically inspired algorithm that provides an adaptive concept for solving optimization problems and designing metaheuristics algorithm (Dorigo & Stützle, 2004; Ferdaus, Murshed, Calheiros & Buyya, 2014). This algorithm is based on an evolutionary approach where the best solution is searched by a group of ants that work together within the colony. The complete solution is built by combining all the individual solutions of each ant which, in another term is known as pheromone deposit, on a chosen solution or path. The strength of pheromone is used by other ants as a reference to choose the most optimized path. An ant will first search for the path by using a probabilistic decision rule that considers the pheromone left on a specific trail over the total pheromone left on all trails. The probability will be relatively controlled by the amount of pheromone and distance between job and resource. The amount of pheromone will continue increasing whenever the trail is chosen by an ant, making it more attractive to the next ant (Ankita & Sahana, 2019). Typically, the evaporation rate is a constant value defined before the execution. However, constant value is not suitable to be used in a dynamic environment where execution parameters may change from time to time. Thus, it is essential to dynamically define the evaporation rate so that it will not be too high or too low. The adaptive evaporation rate formula was proposed by Mavrovouniotis and Yang (2013). If stagnation occurs, the evaporation rate should be increased so that the high intensity of the pheromone trail can be eliminated and eventually will increase exploration. In contrast, if there is no stagnation, the evaporation rate should be decreased gradually so that the optimal solution can last longer.

The concept of solution construction by combining individual solutions is the key criteria for ACO to be widely adopted and adapted in solving job scheduling, load balancing and fault tolerance algorithm in dynamic grid environment (Idris et al., 2017). The ACO itself is dynamic in nature because it is designed to work in dynamic environments (Chowdhury et al., 2019). For instance, in job scheduling problem, ACO is typically used to improve the scheduling decision by considering the size of jobs, capacity of resources and distance between both (Ku-Mahamud & Nasir, 2010; Tiwari & Vidyarthi, 2016). In load balancing aspect, the pheromone trail is an important value to indicate the desirability of constructed paths or resources. According to ACO concept, the higher the pheromone value, the more desirable the path or resource is. This indication is often used to determine potential congestion or stagnation in the system to invoke necessary action to avoid such events (Karimpour, Khayyambashi & Movahhedinia, 2016; Mahato et al., 2019). In terms of fault tolerance, the same concept used to solve job scheduling and load balancing problems is further extended to consider additional aspects to improve fault tolerance capability. This includes the pheromone value that indicates the resource fitness and job scheduling decision during reprocessing stage (Prashar et al., 2014; Rajab & Kabalan, 2016).

The Ant System (AS) was the first member of well-known ACO algorithms proposed by Dorigo, Maniezzo and Colorni (1991), Dorigo et al. (1991) and Dorigo (1992) with the aim to search for the optimal path from the graph constructed based on ants' behavior to seek for a path between colony and food source. The AS was also applied in the Traveling Salesman Problem algorithm proposed by Dorigo and Gambardella (1997a, 1997b). AS consists of three different versions such as ant-density, ant-quantity and ant-cycle. The ants update the pheromone directly when they move from one city to another in ant-density and and-quantity. However, in ant-cycle, the pheromone update is only applied when all the ants completed the tours. Generally, AS consists of two main phases which are solution construction and pheromone update. In solution construction phase, probabilistic decision rule based on pheromone and heuristic values is used to decide the next node that the ants should visit. Then, the unvisited arcs will undergo pheromone evaporation to reduce their attraction factor for the next ants while he visited trails will undergo pheromone update or deposition to increase the attraction factor for the next ants. In the end, an optimal trail will be constructed from source node to the destination node. In short, ants would construct the solution and only follow the pheromone update process which resulted in a dramatic decrease in performance when the size of test instances increased.

The first known improvement of AS is called Elitist strategy for Ant System (EAS) which was introduced by Dorigo et al. (1991, 1996). The improvement was done in terms of providing additional reinforcement to the arcs belonging to the global best tours on top of the standard operations of AS. In other words, the global best tours will receive additional pheromone deposit by the best-so-far ant in every iteration even if the trail is yet to be visited. The main aim of EAS is to allow the ants to find better tour with lower number of iterations.

Another improvement of AS is called rank-based ant system (AS_{rank}) which was introduced by Bullnheimer, Hart, and Straub (1999). In AS_{rank}, each ant is sorted and ranked according to its tour length and the quantity of pheromone to be deposited is weighed according to the rank whereby the shorter the length, the higher amount of pheromone will be deposited. As in EAS, the best-so-far ant will be given priority to deposit more pheromone during iteration. In additional to that, only the best ranked ants and the ant that produces the best-so-far tour are allowed to deposit the pheromone. Both AS_{rank} and EAS produced significant improvement over AS with AS_{rank} achieved slightly better performance than EAS.

To extend the capability of the AS algorithm, Dorigo and Gambardella (1997a, 1997b) proposed the ACS. First, the proposed algorithm uses a more aggressive action choice rule when compared to the AS. Then, it adds the pheromone to arcs that belong to the global best solution and, lastly, it reduces some of the pheromone from the arc upon usage of the chosen path. In terms of the pheromone update process, only the global best ant is allowed to add pheromone after each iteration and the update is applied to the global best path only. This process is known as the global updating rule which reduces the probability of an already visited path being chosen by the next ant in order to increase the exploration probability for a yet to be visited path. However, there are cases where iteration-best solution is also implemented in the ACS.

In ACS, the movement of ants from one node to another is performed using two basic rules which are pseudorandom proportional rule based on exploitation mechanism and exploration mechanism based on probability distribution rule as in AS. The desirability of ants to choose either rule is controlled using fixed variable which is controlled by the user. There are two types of pheromone update introduced in ACS which are local pheromone update and global pheromone update. The local pheromone update is applied when the ant moves from one node to another to reduce the pheromone intensity of the visited arcs by using evaporation concept. This approach is used to reduce the attraction factor of visited arcs to increase the exploration of the next ants. On the other hand, the global pheromone update is applied by the best-ant-so-far to increase the pheromone intensity of the global best path. This action is essential to increase the attraction factor of the arcs for the next ants that select exploitation mechanism to perform the tour (Alobaedy, 2015).

Typically, the ACS focuses on the global-best solution which might lead to poor quality solutions found by ants. To overcome this possibility, Max-Min Ant System (MMAS) was introduced in order to exploit the iteration-best solution while maintaining other criteria set by the ACS (Stützle & Hoos, 2000). This means that each iteration could have a different best solution or similar best solution from previous iterations. Meanwhile, in the ACS, the best solution from previous iterations will be adopted by the new iteration. Additionally, the trail limit is also set within a defined range so that no path has too high or too low a pheromone value. Even though MMAS produced better results when compared with the ACS in certain implementations, the range definition will not be used in the proposed algorithm since the load balancing component will ensure that no resource will ever have too high or low a pheromone value. However, often, ACO consists of a mixture of the AS, ACS and MMAS in order to tackle specific problems. For example, a pheromone update technique could be derived from the ACS while the probabilistic decision rule could be derived from the AS.

Each ACO variants have similarities and differences which are introduced to overcome the disadvantages of previous variants. Table 2.4 summarizes the key differences of each variant.

Table 2.4

Summary of key characteristics for various ACO variants

ACO Variant	**Key Characteristic**
AS	Each individual ant deposits the pheromone on visited arcs during the tour
EAS	Additional reinforcement for best-so-far ant to deposit more pheromone to the arcs
AS_{rank}	Pheromone deposit is controlled based on the rank of tour length for each ant and only best-so-far ant can update significantly large pheromone
ACS	Focus on search experience with global pheromone update and local pheromone update that can be controlled to increase the exploitation or exploration
MMAS	In addition to exploiting the best tour found, range of pheromone is defined to limit the pheromone evaporation to go below the minimum limit and pheromone will be reinitialized once the stagnation is detected

Over the years, ACO has evolved throughout many variants to overcome the limitations of predecessor AS algorithm. Despite many variants of ACO, ACS has been the variant adapted and improved widely by the researchers to cater different application domains and problems such as routing, scheduling, load balancing and fault tolerance. Often, researchers used the term ACO more commonly as the proposed ant-based algorithms consist of combination of multiple variants or only parts of the original variant are adopted and adapted. However, there are also researchers that specifically used the name of the variant as the originality of the variant is fully adopted and adapted with improvements. Despite the inconsistencies in the term, the concept of ACO is still the core of the algorithm.

2.3.1 ACO-based Scheduling and Load Balancing in Grid Computing

ACO-based scheduling and load balancing has been widely implemented in distributed systems such as grids (Bagherzadeh & MadadyarAdeh, 2009; Sharma, Sharma & Dalal, 2014; Prashar et al., 2014; Idris et al., 2017; Kumar & Vengatesan, 2019), cloud (Nishant et al., 2012; Chen & Long, 2019) and cluster (Llanes et al., 2016). The main objectives of scheduling and load balancing in various distributed systems are relatively similar but, in each system, there are specific considerations such as the resource load or capacity in grid, type of resources in cloud, and interaction between cluster heads or within the cluster itself. The adaptability of ACO to consider these specific considerations is one of the main reasons why it remains one of the promising algorithms that can be further enhanced.

Bagherzadeh and MadadyarAdeh (2009) proposed an improved ant algorithm for static grid scheduling with the aim to minimize jobs processing time and balance entire resources in grid computing systems. Every single ant will have its own job to resource the matrix which is also known as the scheduling list. Then, a minimization function will be applied before a probabilistic decision formula is calculated to update the pheromone trail for each job and resource pair. Ultimately, the best solution will be identified. The proposed algorithm was experimented and compared with Opportunistic Load Balancing, Minimum Execution Time, MCT, Switching Algorithm, K-Percent Best, MinMin, MaxMin, MaxStd, Dupplex, and previous ACO algorithm and the results showed that it performed better in terms of minimizing makespan. Alternatively, improved ACO was also claimed to have good load balancing among machines and can be further enhanced by incorporating local search.

Enhanced heuristic function in ACS was proposed by Ku-Mahamud and Alobaedy (2012) to solve stagnation problem in grid computing system. The proposed algorithm differs from the traditional ACS where the new heuristic function is introduced to either increase or reduce the heuristic value based on the quality of the best-so-far solution. The heuristic value will be updated only once on each edge if it is part of the best-so-far edge. The heuristic value update is performed after the global update process that is based on the best-so-far solution is applied after the ant has constructed the solution. The enhancement was claimed to be able to eliminate stagnation problem if the heuristic value is updated multiple times. The improved ACS showed good results when compared with the traditional ACS in terms of makespan and resource utilization.

Improved Auto-Controlled Ant Colony Optimization (IAC-ACO) was proposed by Tiwari and Vidyarthi (2016) to achieve faster convergence of the solution and increase the probability of exploitation around the best ant. IAC-ACO introduces the lazy ant component in balancing convergence and diversification during the searching process. The lazy ant is mutated from the best ant which carries some information that helps to reduce the effort required to construct new path. According to the authors, the lazy ants copy 80% of the path constructed by the best ants. The auto-control mechanism has also been introduced by IAC-ACO to update the heuristic information after each allocation of the task in adapting and updating the changes in the grid system. IAC-ACO showed good performance when compared to the previous auto-controlled ant colony optimization algorithm in terms of computational time.

Load balancing using the ACO and Max-Min technique was proposed by Karimpour, Khayyambashi and Movahhedinia (2016) to prevent stagnation in grid computing systems. The resource manager identifies the best resource based on pheromone value that is stored in the matrix. Once the best resource is identified, global pheromone update will be performed to renew the status of all resources. On top of pheromone-based resource identification, the pheromone value is validated against the threshold to prevent it from decreasing below the minimum limit or increasing beyond the maximum limit. This method increases resource utilization and eventually leads to lower possibility of stagnation during execution. The proposed algorithm was validated with other algorithms in terms of execution time and response time and results showed that it outperformed all the algorithms in both aspects. Despite the promising performance, it was not validated in terms of load balancing aspect.

Hajoui, Bouattane, Youssfi and Illoussamen (2018) proposed a fuzzy hybrid scheduling algorithm by hybridizing Q-learning and ACO algorithms. There are two phases being performed in the proposed algorithm, the first phase utilizes the ACO to perform parallel searches on optimal network links between jobs dispatcher and resources. Once the network link is identified, pheromone will be deposited by the ants. At the same time, jobs dispatchers will also reward or penalize resources based on their condition, powerful or weak, and these values will become the input for Q-learning calculation in the second phase. Q-learning is used to schedule jobs to suitable resources through calculated Q for each machine in which the machine with highest Q is selected to receive jobs. The proposed algorithm was compared with single ACO and Q-learning algorithm in terms of the performance ratio against the load balancing theoretic and it achieved the lowest ratio. However, the experiments were done on small number of resources and jobs, and it was not shown that the proposed algorithm can perform optimally when dealing with a more complex and heterogenous tasks.

Arora and Mehta (2018) proposed resource and task scheduling by combining ACO and round robin scheduling to improve resource management in grid computing. The initial scheduling process is performed using a round robin process that uses time quanta which is tied to each job to handle user requests. Jobs will be assigned to resources to be processed; if processing is not completed within the allocated time, the remaining incomplete job will undergo another round of round robin process. This process will continue until all the jobs are completely processed. The round robin process is simple in nature, leading to less overheard during the scheduling process. Once all the jobs execution is completed, the ACO process will take place to update the pheromone of each resource for the next batch of execution. The proposed algorithm was validated against the non-heuristic load balancing algorithm and the results showed that it achieved lower execution time and resource costing.

Notwithstanding the performance, the experiments were carried out on small number of resources and jobs, and may not be efficient when processing large jobs that are prone to unexpected delays which will eventually lead to frequent incomplete processing within the allocated time.

Mahato et al. (2019) proposed a hybrid swarm intelligence algorithm known as load balanced transaction scheduling based on CS-ACO in solving the scheduling and load balancing problem in grid computing. The proposed algorithm first applies CS algorithm to find the optimal assignment of nodes to one of the clusters by considering the load. The cluster with minimum cost will be selected as the best cluster to undergo the steps in ACO algorithm to perform the load balancing which includes solution construction by ants, pheromone update and daemon actions. Combination of these two algorithms can control the distribution of jobs in the system. Experimental results showed that the proposed algorithm outperformed other algorithms in terms of throughput, makespan, miss ratio and load balancing speedup. However, by focusing on balancing the load on specific cluster, the resource utilization is being disregarded despite the solution would lead to the lowest makespan.

The trust-based resource selection approach based on ACO was proposed by Kumar and Vengatesan (2019) to overcome the local optima problem of ACO. The trust factor is incorporated in the heuristic information to determine the weight of attraction of each resource which will influence the amount of pheromone deposited by the pheromone updating rule at a particular resource. At first, the solution construction is performed using ACO until convergence happens. When convergence happens, a population of fireflies will be spawned and the fitness of all fireflies will be evaluated and ranked accordingly. This process will continue until maximum iteration is found which will produce the optimal result. The proposed algorithm resulted in a slight decrease of makespan and performance improvement. Since the proposed algorithm takes output of one algorithm as input for another algorithm, overhead may occur due to intensive solution construction performed by two algorithms consecutively.

Based on the related works discussed, many researchers have used the ACO approach in solving scheduling problems. The initial pheromone value is used to define the fitness of resources while the pheromone update mechanism is used to balance the load to overcome stagnation. Further exploration is needed in order to incorporate effective scheduling, load balancing and fault tolerance using the ACO approach, in grid computing specifically.

2.3.2 ACO-based Grid Fault Tolerance

Trust-based ant colony optimization (TACO) for grid resource scheduling was proposed by Wenming et al. (2009) whose research aimed to minimize the completion of jobs, balance all the workload on available resources and, at the same time, introduce the resource-oriented trust mechanism to handle the resource failure problem. The initial pheromone value for resource selection process considers the characteristics of each resource such as number of processors, processing power, communication capability, disc capacity and trustworthiness of resources. The trustworthiness factor depends upon the job processing status where the trustworthiness factor will increase if the job successfully processes the submitted job and vice versa if the job processing fails. The resource with high trustworthiness value will be selected to process the submitted job. In terms of action to be taken after job processing failure, the rescheduling mechanism was proposed where the grid system will re-append a failed job into the job queue to be processed by other available resources. Local and global pheromone updates are also included in this algorithm to solve load balancing problems. Experimental results showed that the proposed algorithm performed better in terms of completion time and number of successful processing jobs when compared with Min-Min algorithm. However, the proposed algorithm did not consider the characteristics of submitted jobs during the resource selection process and the trustworthiness means that there is a possibility that the same resource will be chosen in the next cycle if its initial pheromone value remains high as compared to other resources which could eventually expose the next execution cycle to failure. However, this problem could be addressed by extending the trustworthiness determination to also consider resource suspension so that the resource can be marked as unavailable for a defined cycle to allow it to go through the recovery process such as rebooting and cache clearance. Furthermore, the performance of the proposed algorithm was not compared with the traditional ACO algorithm, thus making it difficult to validate the effect of proposed steps against the traditional ACO algorithm.

A study by Modiri et al. (2011) proposed a new algorithm to manage fault in grid computing by combining the ACO algorithm and DAG. By using the DAG method, all tasks are sorted by their dependency which means that the offspring task may not begin its work until the parent task is completely executed. All the sorted tasks will go through the resource allocation process using ACO where ants will try to find the optimal path for each combination of task and resource. Once the resource allocation is done, tasks will be executed according to their sorted order. The local and global pheromone update techniques were used to balance the system load. The proposed algorithm was compared with the Heterogeneous Earliest Finish Time and Critical Path on a Processor algorithms and the results showed that the execution time is significantly improved in addition to better task distribution to all resources. Even though the proposed algorithm tried to increase fault tolerance throughout effective resource allocation, it does not cover the recovery process when a fault occurs. Effective fault tolerance should consider minimizing the occurrence of faults as well as ways to ensure failed tasks are also executed completely.

Hybrid ACO with GA was proposed by Mandloi and Gupta (2013) in order to overcome the uncontrolled nature of the metaheuristic of ACO which could degrade the performance of grid allocation. GA is used to choose whether to increase or decrease pheromone update parameters in ACO. At first, ants will randomly select resources to be assigned into subsets. Then, each subset will be evaluated to find the lowest estimated error, following which it will be sorted in an ascending order. The best subset will be used to execute tasks in each iteration and the pheromone trail for the chosen subset will be updated in each iteration. Resources within the best subset will have a high chance of being selected in the subsets of the next iteration. The experimental results showed that the proposed algorithm increases the job completion rate and reduces the job failure rate as compared to traditional ACO and particle swarm optimization. However, the proposed algorithm can be further upgraded by considering the load balancing aspect and the way to handle job failure when it happens.

The tentative Ant Colony algorithm was proposed by Sharma, Sharma and Dalal (2014). Typically, ACO focuses on pheromone updates at the path the ants traverse.

However, the proposed algorithm focuses on pheromone updates at the resource in which the update is done based on the task's execution status. If the execution is completed successfully, an encouragement argument will increase the pheromone value which will give the resource a higher possibility to be chosen by the next ant. However, if the execution fails, a punishment argument will decrease the pheromone value making it less desirable for the next ant to choose. Both encouragement and punishment arguments will ensure the best resource is assigned to execute a specific task. Results showed that the proposed algorithm performed better in terms of low processing time and low processing cost when compared with the random resource allocation algorithm. Nevertheless, it does not include a strategy to resubmit or reprocess failed tasks even though it checks for execution status.

Fault tolerance ACO (FTACO) using the checkpoint in grid computing was proposed by Prashar et al. (2014) to solve fault and load balancing problems by finding the optimal resource as well as detecting the occurrence of failure during job execution. At first, threshold level of nodes is declared which will be used to control the load of resources. The selection of nodes is based on the resource load whereby if the load is lower than the threshold level, the resource will be assigned with tasks and the execution will be managed by the checkpoint manager. A component called the fault index manager, which is connected to the checkpoint manager, is introduced to record the failure history that is used as a reference in the next job assignment. Fault index will be decreased upon job completion or increased upon job failure. In addition to the checkpoint manager, part of execution outputs or known as checkpoints are stored in checkpoint server which are retrievable upon failure. When failure occurs, failed tasks will be rescheduled to alternative optimal resources using the checkpoint technique from the last saved state instead of from the beginning. In terms of the load balancing aspect, tasks will have a higher possibility of being assigned to resources with a low workload. The workload is indicated by the pheromone value of each resource which will continuously be updated in every checkpoint call. Although the proposed algorithm looks promising, it is simply a conceptual algorithm which has not been developed and validated to prove its claimed performance.

The ant-based dynamic load balancing algorithm was proposed by Rajab and Kabalan (2016) in which lower and upper thresholds were introduced to determine the load status on resources. In the proposed algorithm, the pheromone is associated with the resource instead of the path, as in traditional ACO. The task assignment process considers the resource with the highest pheromone value to be assigned with the task, followed by pheromone decrease once the task is allocated. When the task execution is successful, the pheromone will be increased and if the task execution is not successful, the task will be added back into the task queue. After the task execution is done, regardless of success or failure, the imbalance of load in resources will be checked to determine whether tasks should be migrated to an underloaded resource or retained at the current resource. The proposed algorithm outperformed randomized algorithm in terms of execution cost and makespan. Despite having the mechanism to balance the load, the proposed algorithm only considered each task as a single task whereby if it fails, it will be reprocessed from the initial state which could lead to longer makespan should the same task keep failing due to system instability.

An improved ACO algorithm with fault tolerance (ACOwFT) was proposed by Idris et al. (2017) that combines checkpoint and resubmission techniques. At first, the jobs are submitted to the scheduler handler which is responsible to get the list of available resources from GIS using gridlet dispatcher and get current load from the resources pool. At the same time, fault index that is maintained by the fault index handler during execution is also retrieved and information is forwarded to the gridlet dispatcher. Once resource information is available, resources load and fault index are will be used to identify which resource is ready to be assigned with jobs. Once identified, jobs will be submitted to the checkpoint 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 job processing time and increase throughput. The checkpoint handler interacts with a scheduler to perform unconditional job scheduling that includes both initial submission and resubmission after failure. ACOwFT was inspired from ACO without fault tolerance by Moallem (2009) in which it was reimplemented with additional fault injection mechanism for validation purpose. In reimplemented ACO without fault tolerance algorithm, the faults are injected and standard rescheduling process which is based on the resource load will be invoked without checkpoint mechanism. When compared with ACO without a fault tolerance algorithm (ACO), the results showed that the proposed algorithm reduces makespan, increases throughput and average turnaround time. Despite having good performance, the consideration of resource load alone is believed not to be an effective method to determine the resource fitness and may lead to a higher chance of execution failure.

Garba et al. (2020) proposed an enhanced checkpointing system that dynamically controls the checkpoint interval based on failure rate, response time and number of checkpoints per individual job. The proposed algorithm was enhanced from Idris et al.

(2017) in terms of replicating checkpoint states to other resources in addition to having dynamic checkpoint intervals calculation. The benefit of replicating checkpoint states to other resources is that whenever the checkpoint manager fails to retrieve the checkpoint state from the failed resource, it can be retrieved from other resources. The results showed that the proposed algorithm achieved improvements in terms of makespan, throughput and turnaround time when compared with Idris et al. (2017). However, it was noted that the replication technique requires more efficient memory management to allocate or deallocate replicas together with a higher cost to deploy and maintain the system, and the load balancing aspect was not considered and validated in the experiments.

Based on all the related works reviewed, ACO is considered as a potential algorithm in grid computing to solve fault problems. Several approaches have been identified to provide fault tolerance such as checkpointing, job resubmission, resource trustworthiness amongst others. However, out of all approaches, job resubmission with resource trustworthiness and suspension seems to be an approach that can be further explored to improve the fault tolerance aspect without disregarding the performance as well as being able to adapt to dynamic grid environments.

Table 2.5 shows the summary of ACO-based fault tolerance algorithms. Algorithms proposed by Modiri et al. (2011), Mandloi and Gupta (2013) and Sharma, Sharma and Dalal (2014) only applied fault avoidance technique by reducing the possibility of faults through improved scheduling process which is claimed to directly control the fault in the system. On the other hand, algorithms proposed by Wenming et al. (2009), Prashar et al. (2014), Rajab and Kabalan (2016), Idris et al. (2017) and Garba et al.

(2020) possess several fault tolerance techniques that do not simply focus on the scheduling process but, also, the techniques to handle faults during runtime. The application of fault tolerance techniques is important as it ensures that the system can still operate in faulty conditions with minimal impact to the jobs submitted by the user. Techniques such as job migration and job retry are the most popular techniques as it allows failed job to be resubmitted to the queue for reprocessing until all the jobs are completely processed.

Table 2.5

Summary of ACO-based fault tolerance in grid

			Fault Tolerance Technique						
Author	**Proposed Algorithm**	**Additional Issues Addressed**	Fault Avoidance	Job Migration	Checkpointing	Job Retry	Job Replication	Penalty/Incentive	Resource Suspension
Wenming et al. (2009)	TACO	Scheduling and load balancing	√	X	X	√	X	√	X
Modiri et al. (2011)	ACO algorithm and DAG method	Scheduling and load balancing	√	X	X	X	X	X	X
Mandloi & Gupta (2013)	Hybrid ACO with GA	Scheduling	√	X	X	X	X	X	X
Sharma, Sharma & Dalal (2014)	Tentative ACO	Scheduling	√	X	X	X	X	√	X
Prashar et al. (2014)	FTACO	Load balancing	√	√	√	X	X	X	X
Rajab & Kabalan (2016)	Ant based dynamic load balancing algorithm	Scheduling and load balancing	√	√	X	√	X	√	X

Idris et al. (2017)	An improved ACO algorithm with fault tolerance	Scheduling and load balancing	√	√	√	√	X	X	X
Garba et al. (2020)	Enhanced checkpointing system with replication	Scheduling and load balancing	√	√	√	√	√	X	X

2.4 Summary

Grid computing is an important application domain due to its primary focus on data processing which is critical and requires robust system. The most recent application domain such as cloud computing emerged from the grid computing but with primary focus on providing services such as data sharing, storage, software as a service, platform as a service and infrastructure as a service, that also includes high performance computing offered by the grid computing system. In order to solve failures in grid computing, many researchers have proposed fault management algorithms which consider the processing time of each job and utilization of each resource. Based on the previous research conducted, ACO is proven to be the most promising algorithm that has been successfully used in solving scheduling, load balancing and fault problems in grid computing. Nevertheless, there remain areas of improvement in terms of rescheduling and job migration algorithms in addition to balancing the load using ACO techniques such as initial pheromone value calculation, and local and global pheromone update.

Regardless, in any application domain, fault tolerance algorithms have typically evolved from scheduling algorithms in which the scheduling process is further extended to adapt to the faulty environment. There are several important aspects in scheduling algorithms that should be considered when extending the capability to provide fault tolerance such as execution time, throughput, load balancing and latency. Thus, it is important to consider these aspects in fault tolerance algorithms in addition to execution success rate so that the scheduling process can perform at close to optimal level despite overhead caused by the fault tolerance capabilities. Table 2.6 shows the list of performance evaluation metrics used in previous works related to job scheduling, load balancing and fault tolerance in grid computing.

Table 2.6

Summary of performance evaluation metrics for fault tolerance algorithms in grid

		Performance Evaluation Metric						
Author	**Proposed Algorithm**	Job Completion Time / Execution Time	Success Rate / Failure Rate	Processing Cost	Throughput	Turnaround Time / Makespan	Latency	Resource Utilization / Load Balancing
Wenming et al. (2009)	TACO	√	√					
Modiri et al. (2011)	ACO algorithm and DAG method	√						
Qureshi et al. (2011)	Hybrid fault tolerance techniques				√	√	√	
Keerthika & Kasthuri (2011)	Fault tolerance time to release		√					
Balasangameshwara & Raju (2012)	Fault tolerance hybrid load balancing strategy	√					√	√
Bawa & Singh (2012)	Application checkpointing based fault tolerance technique	√						

Amoon (2013)	Fault tolerance checkpointing system	√	√		√			
Keerthika & Kasthuri (2013)	BSA		√			√		
Mandloi & Gupta (2013)	Hybrid ACO with GA		√					
Sharma, Sharma & Dalal (2014)	Tentative ACO	√		√				
Rathore (2015)	Priority-based scheduling with load balancing	√		√				√
Rathore & Chana (2015)	Threshold-based hierarchical load balancing technique					√	√	√
Rajab & Kabalan (2016)	Ant based dynamic load balancing algorithm	√		√				
Singh (2016)	Failure aware scheduling algorithm based on incremental checkpoint scheme						√	
Singh & Bawa (2016)	Proactive fault tolerance algorithm for job scheduling	√	√	√				
Abdullah, Ali & Haikal (2017)	Hierarchical organization model for computational grid	√					√	
Haider & Nazir (2017)	Hybrid fault tolerance scheme based on proactive and reactive approaches			√	√	√	√	
Idris et al. (2017)	An improved ACO algorithm with fault tolerance	√			√	√		
Goswami & Das (2018)	Fault tolerance nearest deadline first scheduled	√				√		

Ahuja & Banga (2019)	Resubmission-based fault tolerance approach for jobs scheduling	√					√
Garba et al. (2020)	Enhanced checkpointing system with replication	√			√	√	

As shown in the table, job completion time or execution time is mostly used to validate the performance of fault tolerance algorithms, and followed by turnaround time or makespan, latency, success or failure rate and throughput. The details of each performance metrics validated in this research are presented in Chapter 3.

CHAPTER THREE

FRAMEWORK AND METHODOLOGY

This chapter covers the framework and methodology that have been used throughout the research process for Dynamic ACS-based Fault Tolerance with Suspension (DAFTS) algorithm. It starts with Section 3.1 which illustrates the proposed research framework. This section also explains in detail about the methods used in each research stage and expected outputs respectively. Section 3.2 discusses the grid simulation model which includes the architecture of grid computing, system model and application model. Then, the evaluation methodology is covered in Section 3.3 and followed by the performance metrics used in validating the proposed algorithm which are covered in Section 3.4 and 3.5. Lastly, Section 3.6 summarizes the whole framework and methodology.

3.1 Research Framework

There are four main phases in conducting this research which is based on the experimental research framework. The first phase is to investigate and identify the fault tolerance techniques to be used in the DAFTS. This phase is the most critical to identify the suitable fault tolerance techniques such as job resubmission using checkpoint and resource suspension. Secondly, resource assignment and job scheduling are further enhanced to dynamically consider evaporation rate, resource execution history as well as current pheromone intensity. The third phase is the fault tolerance algorithm improvement which combines the output of the first and second phases, and improvement to the ACS formulae to cater the fault tolerance capability. The last phase is the performance evaluation in which the benchmark algorithms are identified and reimplemented in the same simulation platform, performance metrics are identified, experiments are carried out and results are analyzed. This framework is used because it covers all the required steps to propose a new fault tolerance algorithm, and is easy to use in solving scheduling, load balancing and fault problems in grid computing. Evaluation of the newly proposed algorithm, as adopted by most researchers, was done in a simulated environment called GridSim. Simulation allows users to define parameters and test different scenarios and conditions easily. Simulation also allows other algorithms to be developed in the same testbed environment and executed using standard parameters for a more unbiased evaluation. The results are compared against other algorithms for the same set of performance metrics. This methodology was used by Keerthika and Kasthuri (2011, 2012, 2013), Mandloi and Gupta (2013), Rathore and Chana (2015), Rajab and Kabalan (2016), and Idris et al. (2017).

The research framework of DAFTS in grid computing is presented in Figure 3.1. There are four main phases that drive the implementation of this research which consist of determining the designing fault tolerance techniques identification, resource and job scheduling enhancement, fault tolerance algorithm improvement, and performance evaluation of the proposed fault tolerance algorithm.

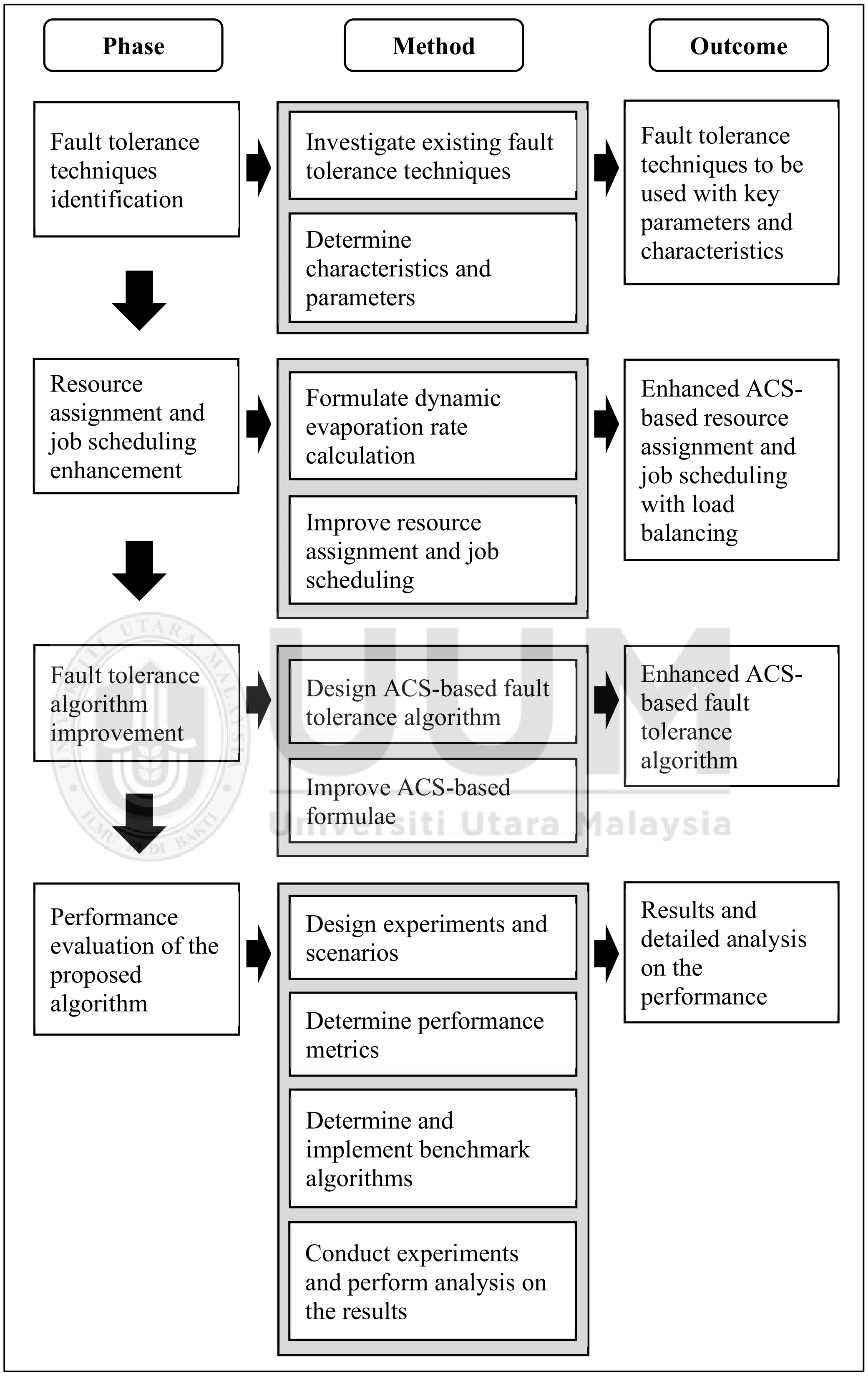

Phase
Method
Outcome
Fault tolerance techniques identification
Investigate existing fault tolerance techniques
Determine characteristics and parameters
Fault tolerance techniques to be used with key parameters and characteristics
Resource assignment and job scheduling enhancement
Formulate dynamic evaporation rate calculation
Improve resource assignment and job scheduling
Enhanced ACS-based resource assignment and job scheduling with load balancing
Fault tolerance algorithm improvement
Design ACS-based fault tolerance algorithm
Improve ACS-based formulae
Enhanced ACS-based fault tolerance algorithm
Performance evaluation of the proposed algorithm
Design experiments and scenarios
Determine performance metrics
Determine and implement benchmark algorithms
Conduct experiments and perform analysis on the results
Results and detailed analysis on the performance

Figure 3.1. Research framework of DAFTS

3.1.1 Fault Tolerance Techniques Identification

Fault tolerance techniques are identified from the list of existing techniques. These techniques include job replication, checkpointing, job resubmission, alternate task, alternate resource, penalty and suspension. After careful consideration and review from previous studies, the job resubmission technique with checkpoint is selected in order to be combined with the penalty and resource suspension technique. Then, applicable parameters and characteristics of chosen techniques are identified in order to be considered in the enhancement phases. The main outcomes of this phase are the fault tolerance techniques applied in the proposed algorithm and their parameters as well as characteristics.

3.1.2 Resource Assignment and Job Scheduling Enhancement

The resource assignment and job scheduling process are enhanced using the dynamic evaporation rate. The dynamic evaporation rate is formulated based on the number of jobs and resources to assign the most optimal evaporation rate which would improve the load balancing aspect. In addition to that, the selection of optimal resources is formulated to consider the highest pheromone and resource availability indicator to ensure that fit resources are utilized to process more jobs as compared to unfit resources. The outcome of this phase is the improved resource assignment and job scheduling that caters the load balancing aspect by having a more controlled resource selection that would not submit new jobs to the suspended resources to reduce the possibility of another failure and to allow the remaining tasks in the resource queue to complete.

3.1.3 Fault Tolerance Algorithm Improvement

The first part of the improvement is to apply all the techniques identified in the first phase and improvement on the resource assignment and job scheduling in the second phase. The second part is to improve the ACS formulae, specifically on the local pheromone update to consider additional aspects such as resource execution history and trust factors. This improvement is essential to better control the pheromone deposited or evaporated at the resource which eventually represents the resource fitness for the ants to perform resource selection during initial state or during reprocessing state. The outcome of this phase is the enhanced ACS-based fault tolerance algorithm that applied suitable techniques with improvement on ACS formulae to not only focus on resource assignment, job scheduling and load balancing, but also on the fault tolerance capability.

3.1.4 Performance Evaluation of the Proposed Algorithm

The proposed algorithm is designed and coded using Java programming language and simulated using GridSim. Experiments and scenarios are designed in order to effectively evaluate the performance of the proposed algorithm against a benchmark algorithm. The results are analyzed to further evaluate improvements in performance and will be reported in the form of diagrams, tables and detailed elaborations.

Several experimental scenarios are conducted such as to measure the effect of the dynamic evaporation rate, incentive and penalty factors and temporary resource suspension. In the experiments, different basic parameters are changed such as failure rate, number of tasks, size of individual tasks, and number of resources. In terms of benchmarking with other algorithms, several measurements are considered such as execution time, latency, throughput, success rate and load balancing. All the parameters and execution characteristics are similar or close to similar as used by other algorithms. This method ensures that the comparison is done fairly and accurately. Several approaches are used in order to obtain execution results from other algorithms. Firstly, the results are obtained throughout simulation of other algorithms. In order to achieve this, a source code or compiled application needs to be available. Secondly, if it is not possible to have the source code or compiled application, the results are obtained from written sources such as a journal or other forms of reliable publications. The last method, which is the least preferred, is to manually code the algorithm based on its pseudocode and execute it. However, this method is very risky because there are conditions or components that may differ with the actual source code or application developed by the original authors which may eventually produce inaccurate or wrong results. Regardless of the method to replicate the implementation of other algorithms, the written source codes are validated against the original results presented in the original works in terms of the value and pattern of the output in table or graph.

Before the experiments to compare with other algorithms are conducted, the parameter tuning experiments were conducted to fine tune the proposed algorithm in terms of effectiveness of dynamic evaporation rate as compared to static evaporation rate, the optimal values for incentive and penalty which are part of trust factor, and effectiveness of suspension technique over without suspension. These experiments are meant to proof that the proposed techniques will improve the performance and also to identify the optimal trust factor which may vary when the proposed algorithm is implemented in different simulation environment, real system and application domain.

The first set of experiments to compare with different algorithms is to validate the effectiveness of the proposed algorithm to adapt with different rates of failure. This is achieved by setting the expected success rate based on pseudorandom algorithm to generate initial resource fitness and failure probability during execution. It is expected that the higher the probability of failure, the lower the execution success rate and throughput, and higher average makespan, average latency and execution time, and reduced load balancing.

The second set of experiments to compare with different algorithms is to validate the how the proposed algorithm behaves when the number of tasks is increased. The increase of number of tasks can also mean that the longer time needed to completely execute all the tasks. Typically, in metaheuristic algorithms, the longer the time to construct the solution, the better the solution will be, and this experiment is dedicated to test out the assumption. It is expected that the increase of the number of tasks will lead to higher execution time, throughput, average makespan, average latency and execution success rate, and better load balancing.

3.2 Grid Simulation Model

Grid simulation toolkit is designed to provide a comprehensive virtual grid platform. In typical grid simulation environment, system and application are the two main components being utilized to cater different experimental scenarios while the basic architecture of the grid system is preserved.

3.2.1 GridSim Architecture

Grid computing generally consists of several main components which are interconnected through Internet and reside in different locations. Due to this fact, grid simulation tools are introduced to allow developers or researchers to develop, test and perform analysis to further improve the environment through new or improved architecture, algorithms, policies and strategies. It is also quite impossible for standalone developers and researchers to be able to own a complete grid computing environment that is managed by multiple parties, involve multiple users and hosted in multiple locations.

The GridSim platform is categorized into several layers for simulating grid environment. The first layer focuses on application, user, inputs and results. Second layer consists of grid resource brokers or schedulers which is responsible to manage the jobs submitted by the user from the first layer. The third layer is where the GridSim toolkit provides all the necessary components to be used for the simulations such as application modeling, resource entities, information services, job management, resource allocation and statistics. The fourth layer consists of event simulation infrastructure that leverages SimJava or Distributed SimJava which is a discreate event simulation library based on Java. The last layer is the collection of virtual machines such as personal computers, workstations, shared memory multiprocessors, clusters and distributed resources (Buyya & Murshed, 2002).

3.2.2 System Model

In the grid computing environment, a set of resources are connected via different communication networks with different speeds. Each resource may have one or multiple numbers of machines and each machine may have single or multiple processing elements. The speed of processor or computational power is defined by the number of cycles per unit time. As the processors in each machine can be heterogeneous, they may have different processing power and fitness.

In the experiments that were conducted, each resource is assumed to consist of one machine and each machine may have one or several processors. The processors in the same or different machines consist of different processing power. A machine in the grid system may also have a local user that uses the machine for other computations. From that point, at any one time, a machine may have a background workload associated with it. This will affect the computational time of the tasks assigned. In order to solve this problem, the GridSim toolkit provides users with the ability to define the background workload according to historical and statistical information for each machine. Each resource has a background load associated that is taken from the average load that the resource has experienced at similar times (such as weekends or working days).

3.2.3 Application Model

In order to develop an application model, it is assumed that the applications which are being run or the tasks which are submitted to the grid system consists of a set of independent tasks with no particular order of execution. The tasks that are submitted consist of different computational times, so that each job also requires a different data transmission time and computation time for completion. The length of each job is presented in MIPS and each job has different input and output size requirements. Tasks in the grid computing system can be classified into one of two categories which consist of a computationally intensive or data intensive task. This research focuses on computationally intensive tasks as it is more common in today's real life applications and the waste of computational power of resources is costlier than their memory (Moallem, 2009). Intensive tasks come in two forms consisting of several tasks with extremely large size or many small tasks that are submitted at the same time. The number of available resources keeps changing throughout the simulation process to replicate the real condition where some resources are not available, or their conditions are not fit to accept new tasks, temporarily.

3.3 Simulation Design and Evaluation Methodology

The proposed algorithm is evaluated in a Java based simulation environment known as GridSim toolkit that provides components for simulating and modeling heterogenous grid environments such as a broker, scheduler, topology, resources, GIS and simulation kernel. The Gridsim toolkit was chosen by many researchers (Patel, Tripathy, & Tripathy, 2016; Ismail et al., 2017) to simulate and evaluate their research because it supports modeling of heterogeneous types of resources and resources can be modeled as space shared or time shared mode.

Application that runs in Gridsim toolkit can be simulated with different parallel applications which can be central processing unit or input/output intensive and at the same time can be heterogeneous. The toolkit itself does not have any limit to the number of jobs that can be simulated and allows simultaneous execution. Both static and dynamic schedulers are supported by Gridsim toolkit and network speed between resources can be determined during initialization or hardcoded in the source code. Statistics of all operations are recordable and can be analyzed using Gridsim statistics analysis methods which can be further presented in a more interactive form using Microsoft Excel.

There are a several standard steps suggested by Gridsim team (Buyya & Murshed, 2002) in order to simulate a grid scheduling algorithm using Gridsim toolkit.

i. Create resources with different capabilities and configurations such as PE rating, number of PE per machine number of machines per resource, and pre-defined fitness rating

ii. Create a number of Gridlets (jobs/tasks) with defined parameters such as length, size of input and output

iii. Create a user entity that creates and interacts with the grid resource broker entity to coordinate execution experiment, and also with GIS and resource entities for submitting and receiving processed Gridlets

iv. Implement a grid resource broker entity that performs application scheduling on resources which is part of the allocation policy build in the application package that interacts closely with resource information in the GIS

Before the evaluation with other algorithms is performed, parameter tuning experiments are performed by validating the effect of the dynamic evaporation rate, incentive and penalty, and suspension technique in order to obtain the most optimal parameters. Once the optimal parameters are obtained, the algorithms that are used to evaluate with the proposed algorithm such as TACO (Wenming et al., 2009), FTACO (Prashar et al., 2014), ACO and ACOwFT (Idris et al., 2017) are implemented in the same simulation environment so that thorough experiments can be performed. Each algorithm is executed using the same set of execution parameters for better consistency and an average of 10 executions is undertaken to obtain the final results for each scenario. The standard execution parameters include the following, as listed in Table 3.1.

Table 3.1

Standard execution parameters

Parameters	**Description**
No. of resources	Number of available resources
No. of tasks	Number of tasks to be executed
PE rating	Processing elements rating in millions instruction per seconds (MIPS)
Bandwidth	Network bandwidth
No. of machines / resources	Number of machines per resource
PE per machine	Number of processing elements per machine
Gridlet length	Job length submitted to GIS
File size	Input file size
Output size	Output file size

In addition to standard execution parameters, each scenario has specific parameters which are controlled statically or dynamically. Specific parameters for all experimental scenarios are listed in Table 3.2.

Table 3.2

Specific execution parameters

Scenarios	**Parameters**
Dynamic vs. fixed evaporation rate	Evaporation rate activated or disabled
Incentive and penalty factor	Incentive and penalty range
With and without suspension	Suspension indicator activated or disabled
Comparison on different failure rates	Initial resource fitness range
Comparison on different number of tasks	Number of tasks range

The list of performance metrics for all experiments listed in Table 3.2 are listed in Table 3.3.

Table 3.3

List of performance metrics for all experiments

Scenarios	**Performance Metrics**
Dynamic vs. fixed evaporation rate	Load balancing
	Execution success rate
	Execution time
Incentive and penalty factor	Load balancing
	Execution success rate
With and without suspension	Execution time
	Execution success rate
	Load balancing

Comparison on different failure rates	Execution time
	Throughput
	Average makespan
	Average latency
	Load balancing
	Execution success rate
Comparison on different number of tasks	Execution time
	Throughput
	Average makespan
	Average latency
	Load balancing
	Execution success rate

3.4 Performance Evaluation Metrics

Performance evaluation metrics are chosen based on common metrics used by researchers in evaluating the effectiveness of the grid scheduling algorithm. Makespan (execution time), throughput and turnaround time are adopted from Idris et al. (2017). Average latency and the execution success rate are adopted from Moallem (2009), and load balancing standard deviation for the fault tolerance algorithm is introduced in this research work based on the concept to calculate load balancing by Sheikh, Nagaraju and Shahid (2018).

Makespan or execution time is measured from the moment the first task is submitted to the system, $SubmissionTime_1$ to undergo the scheduling and execution process until the last task *i, $CompletionTime_n$,* is completely processed as shown in equation (3.1). Execution time indicates how efficiently the system can process all the tasks in the queue and is directly related to the throughput, average latency and average turnaround time.

$$ExecutionTime = CompletionTime_n - SubmissionTime_1 \tag{3.1}$$

For efficiency, throughput is used to measure the performance of the fault tolerance system (Khan et al., 2010; Ezugwu et al., 2013; Idris et al., 2017). It defines the number of tasks that can be completed per unit time. The higher the throughput, the more efficient the system is. Throughput (equation 3.2) is calculated by dividing the total number of tasks, *n,* with the total time taken to completely process all tasks.

$$Throughput = \frac{n}{ExecutionTime} \tag{3.2}$$

On the other hand, average turnaround time is used to measure the average time taken by the system to process each individual job. Similar to execution time, the lower the average makespan, the less time is needed to process all tasks. As shown in equation (3.3), it is measured by summing the execution time for each individual task and dividing the result by the total number of tasks *n*.

$$AverageTurnaroundTime = \frac{\sum_{j=1}^{n}(CompletionTime_j - ArrivalTime_j)}{n} \tag{3.3}$$

Average latency is also important to measure the waiting time for each job before being processed by the assigned resource. Lower latency indicates that the system is capable of utilizing all the resources while controlling the idle time for each job. As depicted in equation (3.4), total latency for all tasks is divided by the total number of tasks *n*.

$$AverageLatency = \frac{\sum_{j=1}^{n}(ArrivalTime_j - SubmissionTim\ _j)}{n} \quad (3.4)$$

The ultimate aim for a fault tolerance system is to preserve the execution success rate. Execution success rate is the percentage of completed tasks over the total tasks submitted. The execution success rate (equation 3.5) calculates the total number of successful checkpoints, $CP_{success}$ over the total number of recorded checkpoints (CP_{failed} + $CP_{success}$).

$$SuccessRate = \frac{\sum CP_{success}}{\sum(CP_{failed} + CP_{success})} \quad (3.5)$$

The higher the execution success rate, the better the system. However, the execution success rate needs to be coupled with load balancing to prevent stagnation or imbalanced resource utilization which will eventually lead to inefficient use of resources.

3.5 Load Balancing Measurement for Fault Tolerance Algorithm

The last performance metric is the load balancing which is newly proposed to measure the effectiveness of resource assignment and jobs scheduling to balance the load distribution in the presence of faults. The load balancing is measured by using standard deviation of the actual against the expected jobs assigned to all available resources in the presence of faults. The standard deviation indicates the difference between the actual outcome versus the mean expected outcome. To calculate the load balancing standard deviation, a population standard deviation formula is used as a base formula where the value to be subtracted from mean is the percentage of total tasks processed by a resource and the mean is the resource fitness rate. Equation (3.6) calculates the difference between total tasks processed and fitness rate for each resource to measure the effectiveness of load balancing aspect.

$$\sigma_r = \sqrt{\frac{\sum(X_r - \mu_r)^2}{N}} \tag{3.6}$$

where X_i is the percentage of total tasks executed by resource i, μ_i is the fitness rate of resource i and N is the total number of resources. The lower the standard deviation, the better the 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 for fault tolerance algorithms, but is not intended to measure the load balancing during runtime.

The proposed load balancing standard deviation measurement is not limited to be used in validating ACO-based fault tolerance algorithm in grid computing environment, but it can also be used to measure the load balancing of both non-fault tolerance and fault tolerance algorithms that are not based on ACO and in other application domains as well. However, the aspect being considered may differ such as consideration of load balancing to the routing path, number of jobs assigned to cluster, node selection to perform hop-to-hop packet delivery and type of jobs and resources.

3.6 Summary

The proposed research framework for dynamic ACS-based fault tolerance with suspension consists of four main phases which include to investigate suitable fault tolerance techniques, propose the enhancement to the resource assignment and jobs scheduling that includes dynamic evaporation rate, improve fault tolerance algorithm and ACS-based formulae, and validate the performance of the proposed algorithm.

Firstly, the investigation phase involves reviewing existing fault tolerance techniques such as job resubmission based on checkpoint, job replication, resource suspension, trust factors and job migration. Techniques that are the most effective are selected to be applied in the proposed fault tolerance algorithm.

Second phase involves introduction of dynamic evaporation rate calculation that is based on the number of jobs and resources, and enhancement to the resource assignment and jobs scheduling process which involves consideration of additional aspects such as resource availability indicator and optimized pheromone value based on trust factors and resource fitness.

In the third phase, the techniques identified in the first phase are integrated with the enhancement from the second phase in the ACS-based fault tolerance algorithm. Existing local pheromone update formula is also enhanced to consider trust factors and resource fitness to improve the way the pheromone is deposited to the resource. All these contribute to the new variant of ACS algorithm for dynamic fault tolerance in grid computing which consists of three main processes: initial pheromone value calculation, resource selection process and fault tolerance mechanism. Initial pheromone update is responsible to predict the best resource to process a specific job. This ensures that a job does not go through random assignment which would lead to inefficient use of execution time. The second process is resource selection where it considers the resource availability flag which is also known as the quarantine state of a specific resource. This ensures that the next job in the queue will not be submitted to a recently failed resource and, eventually, allows the resource to recover. The last process is the fault tolerance mechanism which will resubmit the failed job using a checkpoint technique to other available resources, apply a pheromone update and activate the resource availability flag should the resource fail to process the submitted job successfully. After that, the recovered resource will be put back into the list of available resources that are ready to be assigned to jobs.

The last phase covers the evaluation of the proposed algorithms against the benchmark algorithms. Before the evaluation can be conducted, the experiments and scenarios are designed based on previous studies including the performance metrics. Then, the benchmark algorithms are identified and reimplemented in the same simulation environment as the proposed algorithm based on pseudocode, flowcharts, formulae and architecture design by the original authors. This ensures that the validation is performed fairly and systematically to increase the validity of the results.

CHAPTER FOUR

DYNAMIC ANT COLONY SYSTEM-BASED FAULT TOLERANCE WITH SUSPENSION ALGORITHM

This chapter presents the fundamental design and architecture of the proposed algorithm called Dynamic ACS-based Fault Tolerance with Suspension (DAFTS). Two main aspects are introduced which consist of scheduling and fault tolerance capability. In terms of scheduling, the algorithm considers the execution history during the pheromone update process to influence the desirability of selecting a resource. In terms of fault tolerance capability, resubmission based on checkpoint and resource suspension is applied to ensure all failed tasks can be reprocessed successfully, thereby increasing the success rate. The DAFTS workflow is covered in Section 4.1 and load balancing using dynamic scheduling is elaborated upon in Section 4.2. Resource suspension capability is explained in Section 4.3. Section 4.4 covers the pseudocode of DAFTS and concluding remarks are presented in Section 4.5.

4.1 Dynamic ACS-based Fault Tolerance with Suspension

DAFTS is inspired by the concept of an ant searching for the optimal path to the most suitable resource to assign tasks. In typical grid task scheduling, the ant searches for resources with high pheromone value out of all the available resources. In addition to the pheromone, the load of the resource is also one of the criteria used by the ant to search for the optimal resources to assign submitted task. The pheromone update will be performed upon task assignment to a particular resource and the resource will be released once task execution is completed.

This basic concept is further extended for ants to have the ability to perform the researching process during the resubmission process. In addition, the pheromone update technique is further improved as a mechanism to penalize unfit resources so that they become less attractive and to reward fit resources so that they have better possibilities to be assigned with tasks. This approach is expected to reduce the possibility of failure as the task assignment will focus on fit resources instead of unfit resources.

Figure 4.1 illustrates the phases of DAFTS based grid task scheduling with fault tolerance extension as highlighted. DAFTS differs from typical ACO-based fault tolerance algorithm in terms of the application of local pheromone update process and resource suspension technique. For each task, an ant will be generated to perform resource selection based on the resource pheromone value. The initial pheromone value will first be calculated to determine the state of all resources before the first task in queue can be submitted. Selection of the resource will be based on the amount of pheromone value either from the initial pheromone calculation or pheromone update process. Once a resource is assigned with a task, a global pheromone update will be applied by the ant to reduce the amount of pheromone.

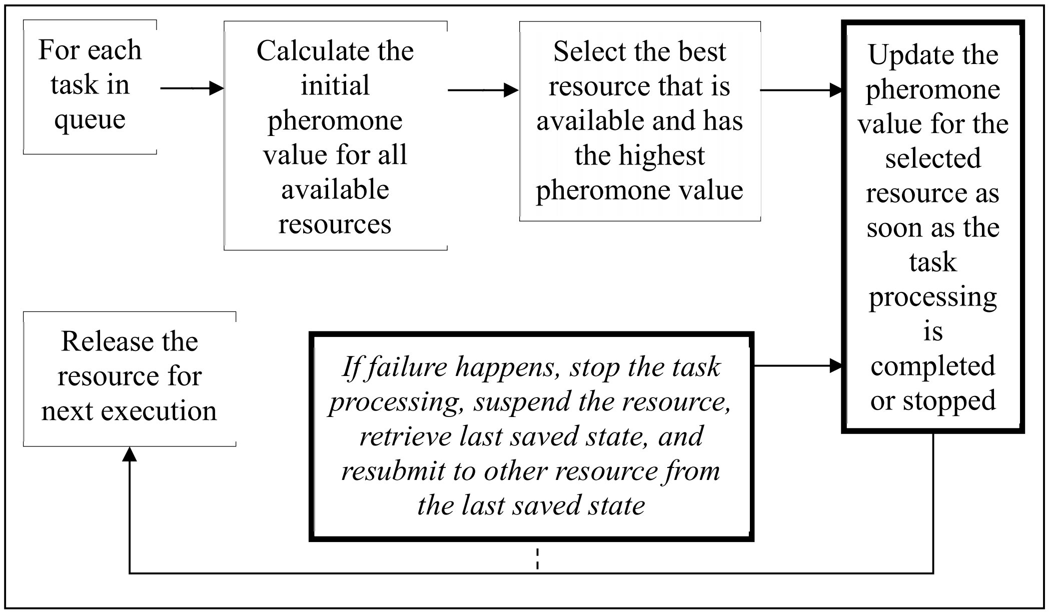

For each task in queue
Calculate the initial pheromone value for all available resources
Select the best resource that is available and has the highest pheromone value
Update the pheromone value for the selected resource as soon as the task processing is completed or stopped
Release the resource for next execution
If failure happens, stop the task processing, suspend the resource, retrieve last saved state, and resubmit to other resource from the last saved state

Figure 4.1. Phases of DAFTS

Each task will be divided into several checkpoints which will be executed in sequence to preserve the authenticity of the output. In each failed checkpoint or complete task execution, the local pheromone update will be applied to reduce or increase the pheromone intensity based on execution history before releasing the resource for the next execution. In case of any failure during execution, the last checkpoint will be resubmitted to another suitable resource and the resource that just failed will be suspended temporarily.

DAFTS is developed in GridSim simulation environment as it provides a close to actual platform without the need to deploy physical resources and involve actual users in carrying out the experiments. Due to complexity of large scale distributed systems, sophisticated simulation tools is demanded to help on analysis and fine tune the algorithm before being applied in the actual environment. In addition to that, it provides flexibility to the developers to modify the parameters and behaviors of various components in the simulation environment to cater different hypothetical problems (Buyya & Murshed, 2002).

4.1.1 Initial Pheromone Value Calculation

Initial pheromone value is calculated after the job is submitted to the grid system. The calculation considers jobs characteristics, resources capacity, estimated transmission time and execution time of the job when assigned to the resource. A higher pheromone value indicates a higher reliability of the resource to process submitted jobs within an estimated time and a lower possibility of job processing failure. Eventually, the initial pheromone value will become the resource pheromone value after the pheromone update.

4.1.2 Resource Selection Process

The selection of the best resource is based on the availability of the resource which is controlled by an availability flag and the highest pheromone value. The zero availability flag indicates that the resource is being suspended temporarily and will not be chosen to process the submitted job. Pheromone value is the key parameter that defines resource fitness. The higher the value, the better the fitness. By considering all these, the selected resource will have the lowest possibility to cause an error and will, thereby, lead to optimization in the grid system. Furthermore, a recently failed resource will have the lowest possibility to be chosen in the next iteration before it goes through the recovery process. Further elaboration on resource selection process that considers both scheduling and load balancing is covered in Section 4.2.

Figure 4.2 illustrates the sequence diagram of the happy flow scheduling process in the proposed algorithm. The grid broker is the source of ant generated for each individual task. This ant is responsible to find the optimal resource in the GIS that stores all information about the resources, current execution state, and checkpoints. Once the optimal resource is identified, a global pheromone update is applied to reduce the pheromone intensity so that it becomes less attractive to the next ant. The task in the queue will be submitted to the identified resource for processing and, once completed, local pheromone update will be applied to either increase or decrease the pheromone of the resource. Once the task is completely processed, the ant will be terminated.

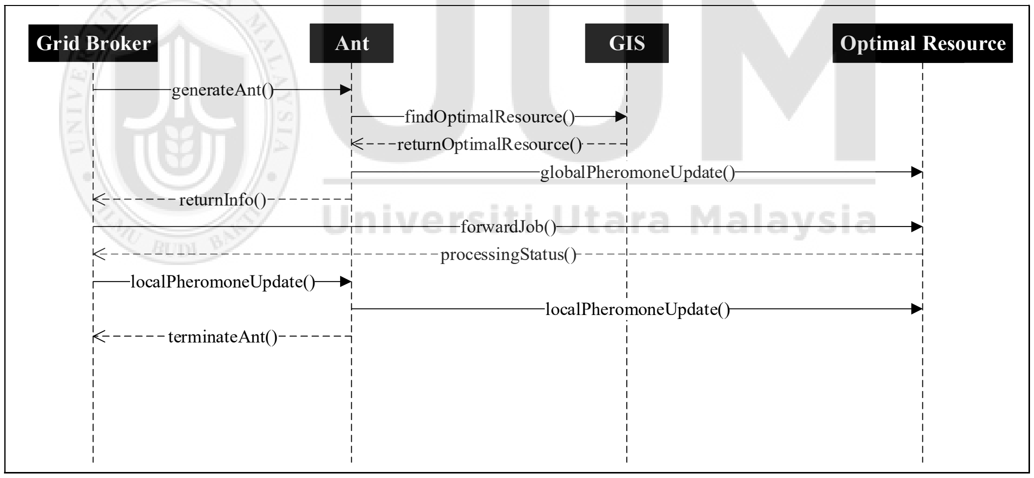


Figure 4.2. Sequence diagram of happy flow scheduling process

4.1.3 Fault Tolerance Mechanism

The fault tolerance mechanism includes the ability of the system to record the checkpoint information at defined intervals, resubmit a failed job from the last saved state to the job queue, apply a pheromone update and activate the availability flag (suspension). When failure happens, the job's execution will be terminated, and the resource will be suspended temporarily for the recovery process. The resource availability flag will continue to be activated after a certain iteration to allow the recovery process and will be reset after meeting the threshold value. All these steps ensure that the resource that failed has time to recover and processing failure could be minimized by temporarily quarantining the resource. Ultimately, the hit rate or job completion rate will be increased even in the presence of processing failure.

Figure 4.3 shows the detailed steps of the fault tolerance scheme in the proposed algorithm. For each task execution, the status of execution will continuously be validated, checkpoint calls will be made, and local pheromone update will be applied based on the status of execution at a point in time. If a task is not completely processed, the remaining task will continue to be executed. Typically, each task execution may take time due to its size, and it is common for failure to happen. When failure happens, the resource that failed to execute the task will undergo a local pheromone update process that invokes the suspension function to temporarily suspend the resource while reducing the pheromone value so that the resource becomes less attractive to the next ant. The resubmission process will be initiated, the last saved checkpoint will be called and put back into the execution queue to undergo the standard scheduling process to find another optimal resource to continue execution until completion. Once task execution is completed, the ant will be terminated. Further explanation on suspension technique is covered in Section 4.3.

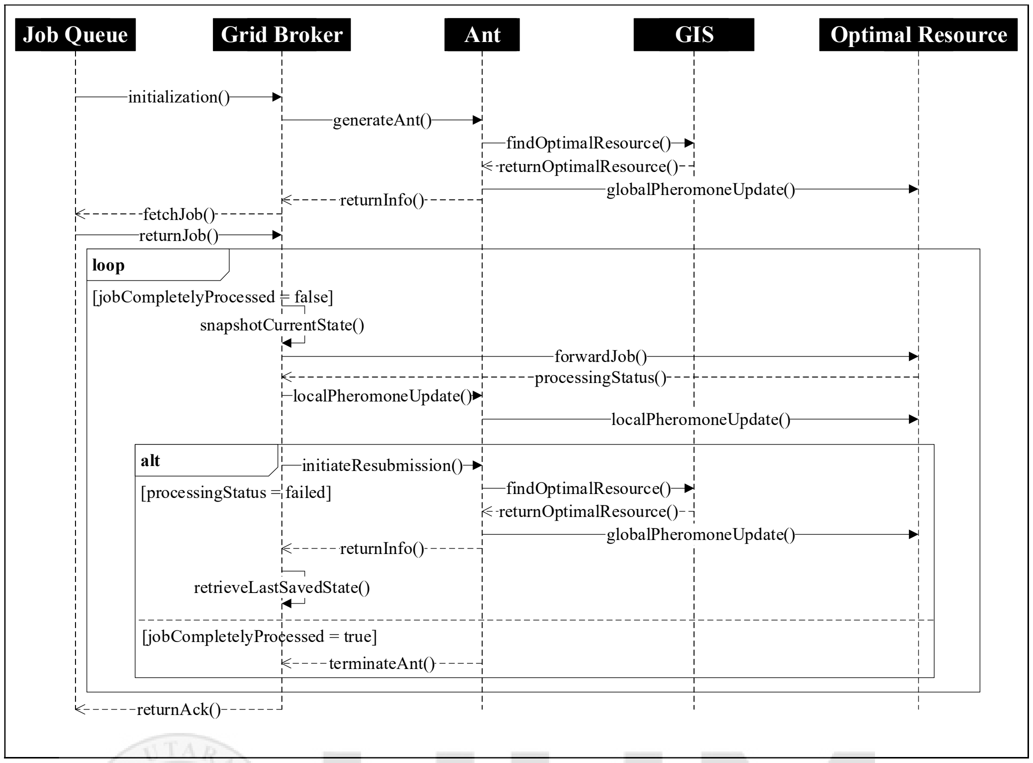
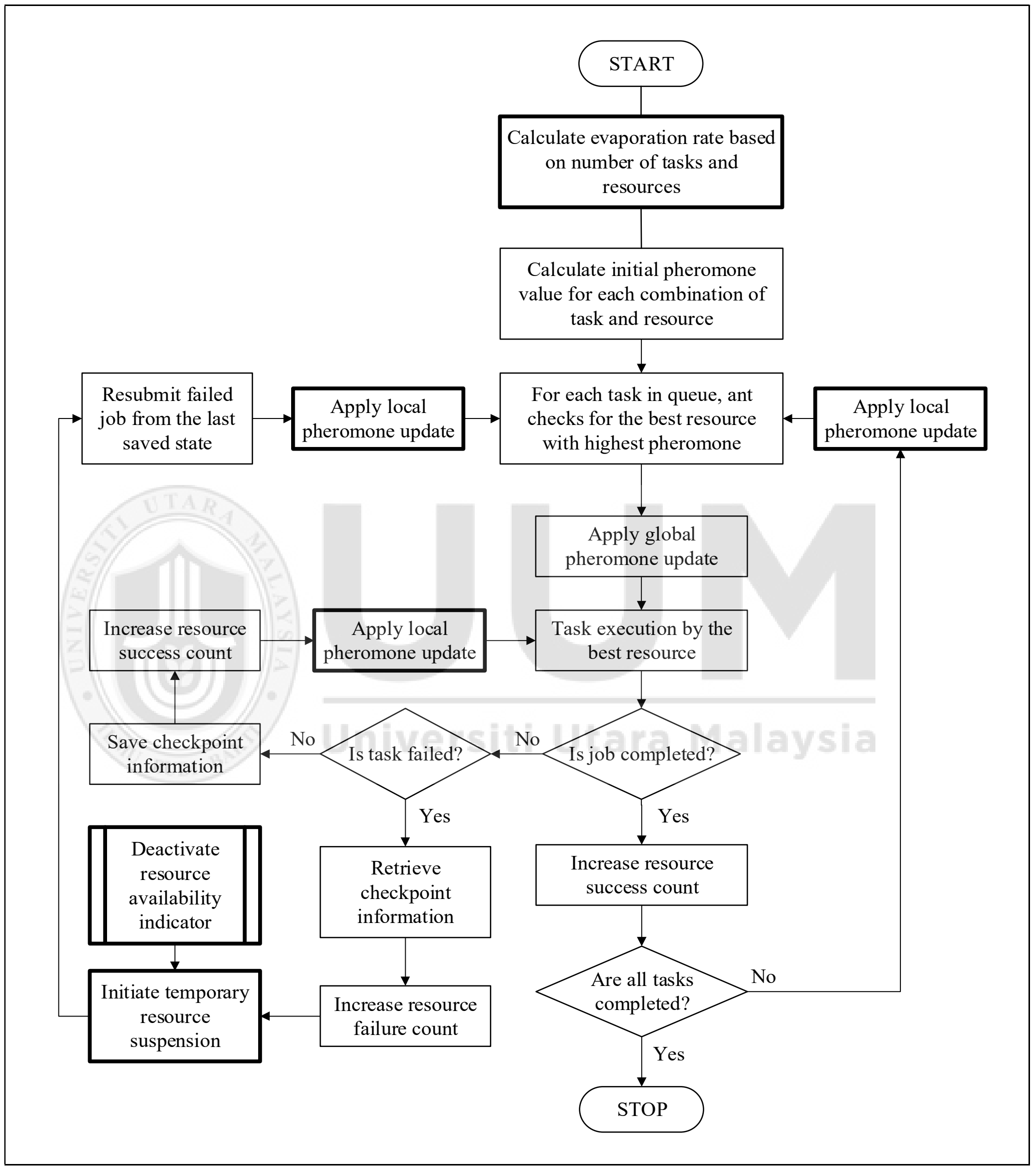


Figure 4.3. Sequence diagram of fault tolerance process

4.1.4 Flowchart of DAFTS

Figure 4.4 illustrates the flowchart of DAFTS with the key contributions of this research bolded. It is the backbone of DAFTS algorithm proposed in this research. Before the first task in the queue can be submitted, evaporation rate will be calculated based on the number of tasks and resources as part of first contribution of this research, and followed by the initial pheromone value to determine the state of all resources. Then, an ant will be generated for each submitted task in the queue to perform resource searching based on pheromone values. The submitted tasks may consists of initially submitted task or task that undergoes rescheduling process. 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 a 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.



START
Calculate evaporation rate based on number of tasks and resources
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 job completed?
Yes
Yes
Deactivate resource availability indicator
Retrieve checkpoint information
Increase resource success count
Are all tasks completed?
No
Initiate temporary resource suspension
Increase resource failure count
Yes
STOP

Figure 4.4. Flowchart of DAFTS

In the event of failure, the scheduler will retrieve the checkpoint of failed task from the checkpoint manager and resource failure count will be increased. After that, the second contribution of this research which is temporary suspension indicator will be invoked to avoid the resource that failed to undergo recovery process or complete the remaining task in its queue. This process is meant to reduce the possibility of another failure by not assigning new task to recently failed resource temporarily and increase resource utilization by assigning new task to alternative resource. The retrieved task will repeat the rescheduling process and will be assigned to an alternative resource after which a local pheromone update with penalty will be applied to the resource that failed to reduce pheromone intensity, third contribution of this research.

In the event of partially successful task execution, the information about successfully completed task will be saved and the checkpoint replica will be removed, and resource success count will be increased. Then, the local pheromone update with incentive will be applied to the resource to increase the pheromone as part of the third contribution of this research. The same resource will continue to execute the remaining part of the task before getting released to process brand new task.

Last but not the least, in the event of complete task execution, the same local pheromone update process will be performed to increase the pheromone intensity to indicate that the resource is fit to receive more new tasks. The whole process will continue until all the submitted tasks are completely processed which means that no task will be left out even in the presence of failures.

4.2 Load Balancing Using Dynamic Scheduling with Checkpointing

During the initial task submission, each resource should have pre-defined parameters such as processor speed, current load, 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 (4.1):

$$PV_{rj} = \left[\frac{S_j}{bandwidth_r} + \frac{C_j}{MIPS_r(1-l \qquad _r)}\right]^{-1} \tag{4.1}$$

where S_j is the size and C_j 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, subsequently, it is considered as a resource pheromone value. The tasks to be processed may come by batch which means that reinitialization will be performed to feed the recently arrived tasks into the current queue. Since the initial pheromone value is calculated for each combination of resource and task, this information is stored in a PV_{matrix} as follows (4.2):

$$PV_{matrix} = \begin{bmatrix} PV_{1,1} & PV_{1,2} & PV_{1,n-1} & PV_{1,n} \\ PV_{2,1} & PV_{2,2} & PV_{2,n-1} & PV_{2,n} \\ PV_{m-1,1} & PV_{m-1,2} & PV_{m-1,n-1} & PV_{m-1,n} \\ PV_{m,1} & PV_{m,2} & PV_{m,n-1} & PV_{m,n} \end{bmatrix} \tag{4.2}$$

where n is total number of tasks and m is total number of resources. PV_{matrix} is a logical form of grid topology whereby an ant would move from one index to another to find the best resource for task assignment. It is assumed that all the resources are interconnected which means that if the task if assigned to a specific resource, it can be migrated 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 the 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 (4.3) 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), $\Delta\tau_{rj}=0$.

$$\tau_{rj} = (1-\rho)\cdot\tau_{rj} + \rho\cdot\Delta\tau_{rj} \tag{4.3}$$

ρ is the evaporation rate that is dynamically controlled by using the following formula (4.4) with *m* and *n* as the total number of resources and tasks respectively:

$$\rho = \left[\left(\frac{n}{m}\right)^{2}\right]^{-1} \tag{4.4}$$

Task assignment will continue while the previously assigned task is being executed. However, if the execution is not successful, the task will be resubmitted from the last saved checkpoint to another suitable resource. On the other hand, the checkpoint information will be recorded during the execution for each task being executed and this information is also used to update the execution history table for each resource. The checkpoint mechanism is applied by splitting a big task into several small tasks which will be submitted sequentially. In DAFTS, the checkpoint interval is set to 5 splits per individual task. This can be adjusted according to the size of task, the bigger the task, the higher number of checkpoint interval will be. When the task is submitted to a specific resource, the checkpoint manager will be responsible to control the feed of every small task by keeping a replica before submitting it to the identified resource and if failure status is received, the small task submitted previously will be resubmitted to the other resource. And if the success status is received from the executing resource, the small task will be removed from the queue.

As shown in Figure 4.5, a large task T1 is divided into 5 small tasks. Based on the information received from the scheduler, the checkpoint manager will set its parameter to assigned T1 to resource B. Then, as in Figure 4.6, the last small task $T1_e$ will be fetched by the checkpoint manager to be submitted to resource B. Before the submission begins, a replica will be created and saved in checkpoint manager's memory. On the other hand, Figure 4.7 shows the event of processing failure by resource A and Figure 4.8 shows that the replica of task $T1_e$ is resubmitted to resource A and followed by Figure 4.9 that shows the replica is removed when it is completely processed.

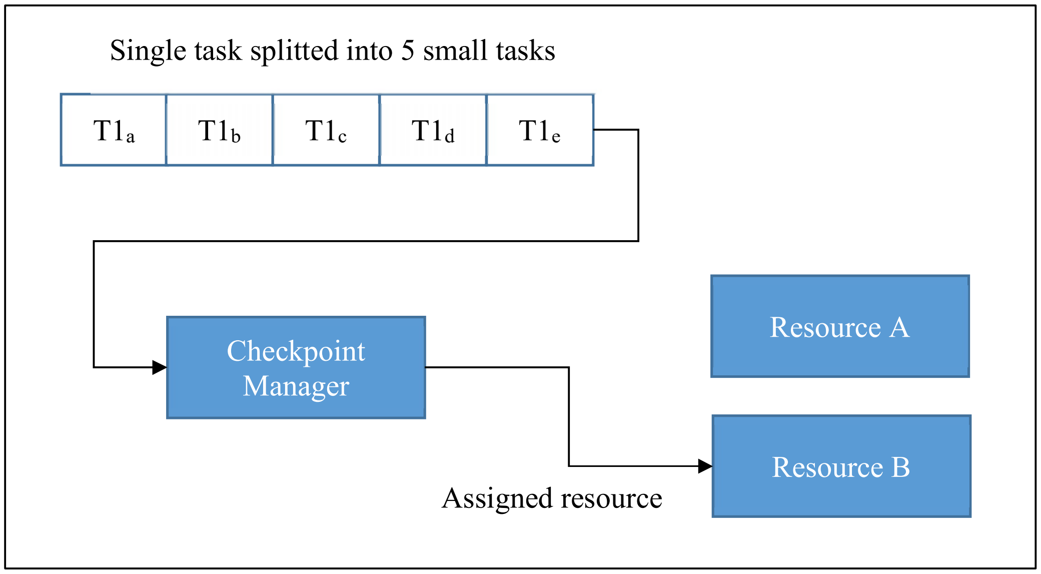


Figure 4.5. Initial state of single task splitted into multiple small tasks flows through checkpoint manager

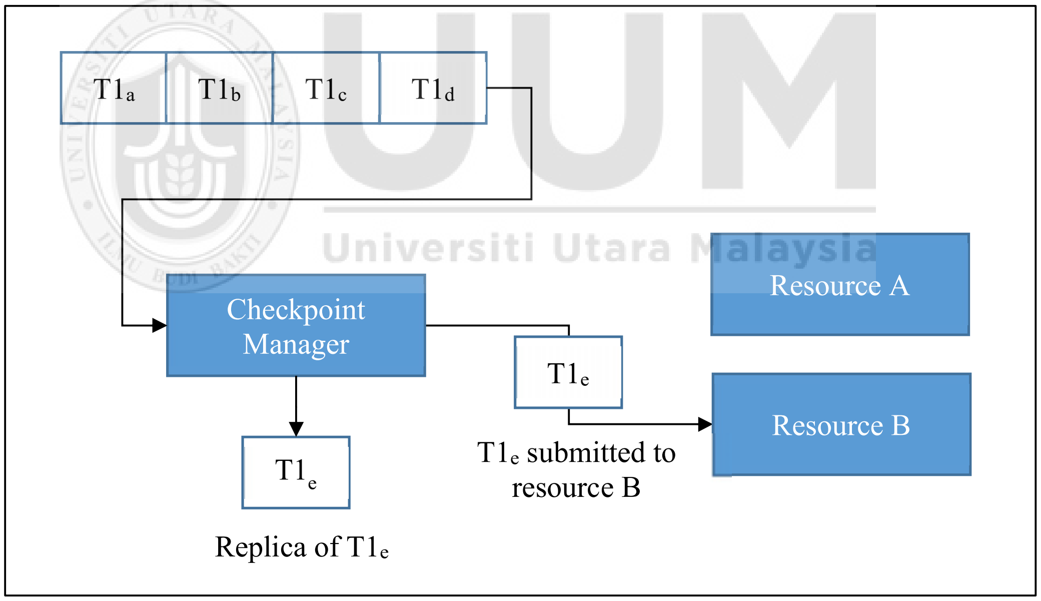


Figure 4.6. Small task submitted to assigned resource and replica saved in the memory

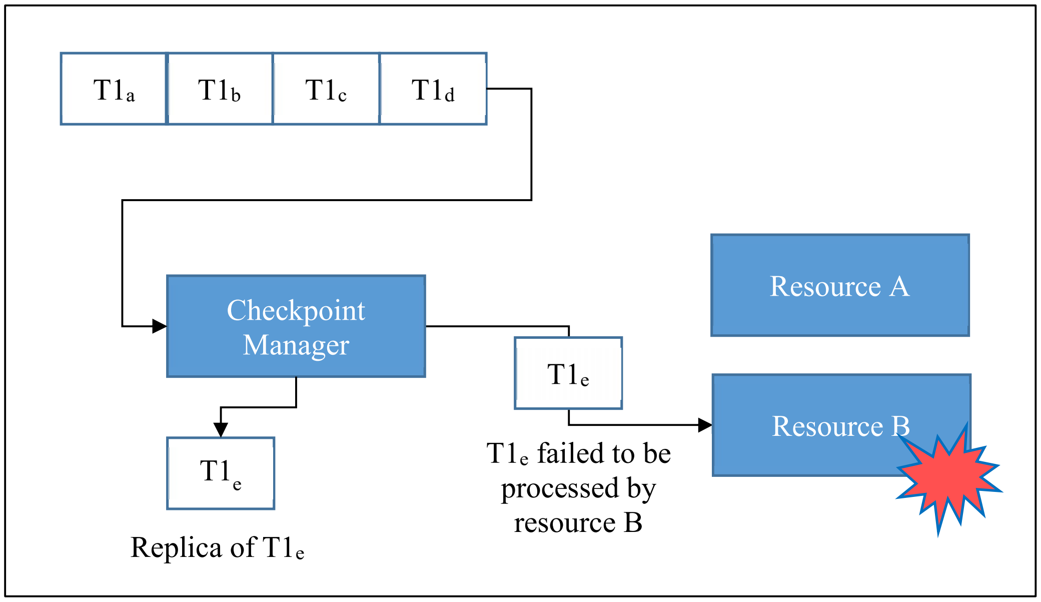


Figure 4.7. Small task failed to be processed by assigned resource

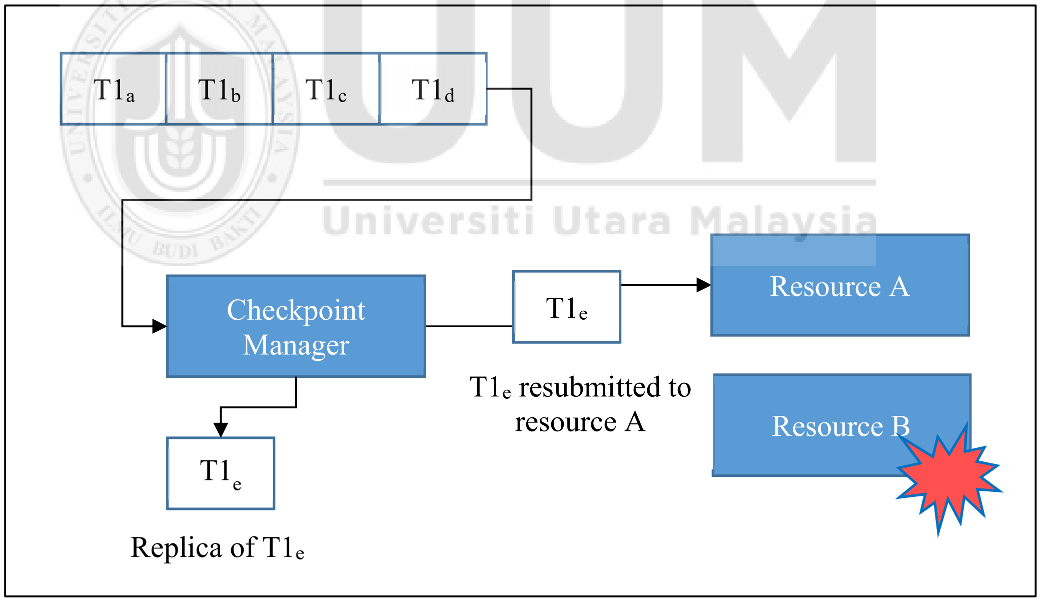


Figure 4.8. Replica is retrieved and submitted to alternative resource

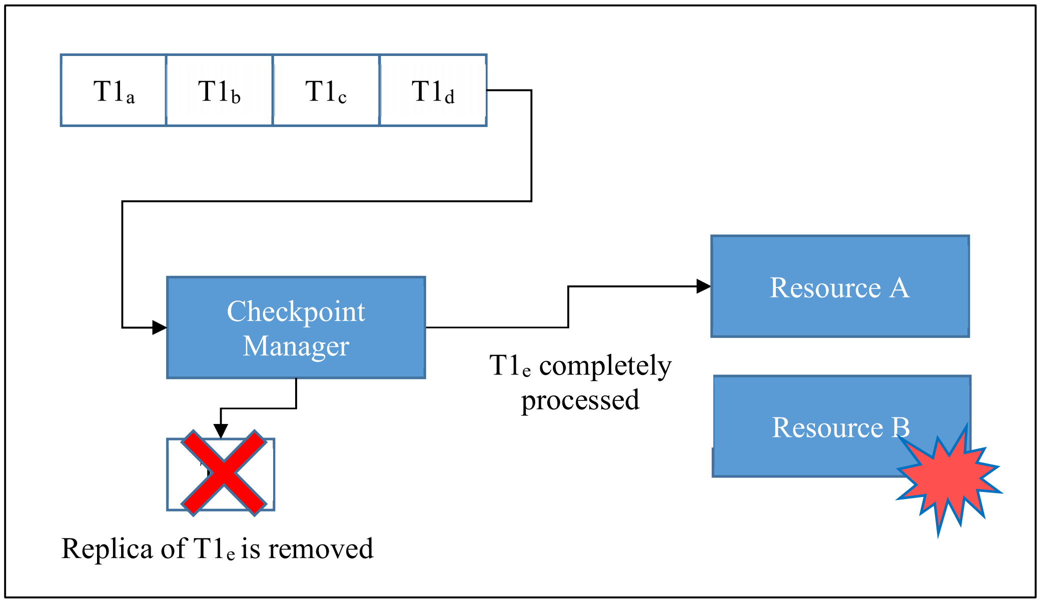


Figure 4.9. Replica is removed after successful processing

In every checkpoint, another round of local pheromone update (4.5) will be applied to reduce more pheromone values by considering the execution history to influence the increment or reduction of pheromones; the success status would increase the pheromones, while the failure status would reduce more pheromones.

$$\tau_{rj} = (1-\rho)\cdot\tau_{rj} + [\rho\cdot\tau_0(R_H)]^T \qquad (4.5)$$

τ_0 is the initial pheromone value of resource r, τ_{rj} is the current pheromone intensity for resource r and task j, T is the trust factor defined by either task completion (T = 1.5) or task failure (T = 1.0) while $R_H(i)$ is the average weighted execution history of resource r and calculated by (4.6):

$$R_H(i) = \begin{cases} R_T(i) = \frac{CP_{success}}{(CP_{failed}+CP_{success})}, i = 0 \\ (1-\alpha)\cdot R_T(i) + \alpha\cdot R_H(i-1), i > 0 \end{cases} \qquad (4.6)$$

where $R_T(i)$ is the total execution history of resource r 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 which 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.

The execution history (defined as resource fitness) is extended to the existing local pheromone update formula and will be used to influence pheromone evaporation or deposition in each resource based on execution status. The better the execution history, the lower the evaporation of the pheromone. This approach is expected to effectively balance the load assigned to each resource so that the resources with good execution history will be assigned with more jobs as compared to resources with bad execution history.

4.3 Temporary Resource Suspension

As covered in Section 4.1, execution history is equivalent to resource fitness where it gives the success probability of a particular resource. Eventually, resource fitness $R_H(i)$ will be used to determine the current failure rate (FR) as follows (4.7):

$$FR = 1 - R_H(i) \tag{4.7}$$

After the failure rate is determined, effective failure rate (EFR) will be determined by using failure rate (FR) and failure indicator (F). The failure indicator is controlled by the broker where the value will be either 0 (success) or 1 (failure). EFR is given by (4.8):

$$EFR = F \times FR \tag{4.8}$$

The formula for resource suspension (RS) is as follows (4.9):

$$RS = EFR \times \frac{\sum i}{\sum j} \tag{4.9}$$

where *i* is the number of tasks and *j* is the number of resources. Suspension value represents the number of cycles a resource should be suspended and will be decremented by 1 in every processing cycle until the count reaches 0. Suspension count is directly influenced by the ratio of jobs over resources to limit the possible suspension count that a resource can undergo. For example, if there are 100 jobs with 10 resources available, the ratio of jobs over a resource is 10 and suspension should not exceed this ratio. Otherwise, the resource will never be assigned after failure.

Resource suspension (RS) will be stored as resource information and used to control the resource availability indicator (RAI) using the following logic:

- If RS = 0, then RAI = 1
- Else (RS > 0), then RAI = 0

RAI will be combined with resource pheromone value taken from the PV_{matrix} to decide which is the resource having a high pheromone value and RAI = 1. Should the resource have the highest pheromone value but RAI = 0, using a multiplication formula between the pheromone value and RAI, the calculated pheromone value will become 0. The calculated pheromone value is controlled by RAI and it will not be applied directly in the PV_{matrix}. In a nutshell, the purpose of having more pheromone deduction (penalty) and resource suspension in the occurrence of failure is to allow a failed resource to undergo a recovery process that includes reboot, cache clearance, network restart and manual recovery.

Assume that there are 3 jobs (T1, T2, T3) and 3 resources (R1, R2, R3). This combination would create a matrix of 3 x 3 mapped with mocked up pheromone values as shown below.

$$PV_{matrix} = \begin{bmatrix} R1,T1 & R2,T1 & R3,T1 \\ R1,T2 & R2,T2 & R3,T2 \\ R1,T3 & R2,T3 & R3,T3 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.61 & 0.9 \\ 0.3 & 0.6 & 0.7 \\ 0.1 & 0.5 & 0.67 \end{bmatrix}$$

Then, assume that R3 has recently failed and RAI = 0. When ant tries to search for the resource to process T1, it will result to the effective pheromone value for R3 equivalent to 0 due to multiplication of actual pheromone with RAI = 0. This will lead to the ant selecting R2 because of it has the highest pheromone value.

$$PV_{matrix} = \begin{bmatrix} R1,T1 & R2,T1 & R3,T1 \\ R1,T2 & R2,T2 & R3,T2 \\ R1,T3 & R2,T3 & R3,T3 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.61 & \boxed{0} \\ 0.3 & 0.6 & 0.7 \\ 0.1 & 0.5 & 0.67 \end{bmatrix}$$

After R2 is selected to process T1, it will undergo global pheromone update process which will reduce its pheromone to encourage ants to select other resource for next task in queue.

$$PV_{matrix} = \begin{bmatrix} R1,T1 & R2,T1 & R3,T1 \\ R1,T2 & R2,T2 & R3,T2 \\ R1,T3 & R2,T3 & R3,T3 \end{bmatrix} = \begin{bmatrix} \begin{matrix} 0.5 \\ 0.3 \\ 0.1 \end{matrix} & \boxed{\begin{matrix} 0.49 \\ 0.41 \\ 0.4 \end{matrix}} & \begin{matrix} 0.9 \\ 0.7 \\ 0.67 \end{matrix} \end{bmatrix}$$

After the T1 is completely processed, the affected row will be removed from the *PV_{matrix}*. The PV_{matrix} will continue to be updated as the new batch of jobs submitted to the grid system.

$$PV_{matrix} = \begin{bmatrix} \cancel{R1,T1} & \cancel{R2,T1} & \cancel{R3,T1} \\ R1,T2 & R2,T2 & R3,T2 \\ R1,T3 & R2,T3 & R3,T3 \end{bmatrix} = \begin{bmatrix} \cancel{0.5} & \cancel{0.49} & \cancel{0.9} \\ 0.3 & 0.41 & 0.7 \\ 0.1 & 0.4 & 0.67 \end{bmatrix}$$

4.4 DAFTS Algorithm

The algorithm starts with initialization process where all the static and dynamic simulation parameters are initialized to form pool of resources and tasks as well as basic components within the simulation environments such as grid resource broker, scheduler, topology, resources, GIS and simulation kernel. Once initialized, initial pheromone value will be calculated using Equation 4.1 which eventually produces pheromone level of each combination of tasks and resources. It is assumed that at this point, the tasks are already submitted by the user and available in execution queue. The grid broker will spawn an ant for each task in the execution queue to identify the resource with highest pheromone level in PV_{matrix} and will submit the task to the identified resource. Once identified, pheromone level will be reduced using global pheromone update (Equation 4.3) to increase utilization of other available resources. Throughout the execution process, checkpoints will be recorded by the checkpoint manager that exists in the grid resource broker and will be restored back upon failure to be submitted to other resources. Each failure or success will be recorded in GIS and local pheromone update (Equation 4.5) will be applied to either increase the pheromone intensity upon success or decrease the pheromone intensity upon failure. In the event of failure, resource suspension will also be invoked to temporarily suspend the failed resource to prevent it from being assigned with new tasks temporarily. Finally, once all tasks in execution queue are completely executed, the algorithm will be terminated. Figure 4.10 represents the pseudocode of the proposed DAFTS algorithm where the research contributions are bolded accordingly.

```
Step 1:  Get number of resources
Step 2:  Get number of tasks
Step 3:  Calculate evaporation rate based on number of tasks and resources (dynamic
         evaporation rate)
Step 4:  For each resource, get resource identification, bandwidth, MIPS and load
Step 5:  For each task, get task identification and task size
Step 6:  For each combination of resource and job combination, calculate initial pheromone
         value and store into pv_matrix array
Step 7:  For each task in queue, create an Ant to search for optimal resource in pv_matrix array
           7.1:    Multiply pheromone with resource availability indicator constant
           7.2:    Get highest calculated pheromone
           7.3:    Assign task to the resource
           7.4:    Apply global pheromone update to the resource
Step 8:  While task execution is not complete
           8.1:    If part of task is completed
                   8.1.1:    Increase resource success count
                   8.1.2:    Apply local pheromone update with incentive (trust factor)
                   8.1.3:    Remove task replica from checkpoint manager
           8.2:    If part of task is failed
                   8.2.1:    Increase resource failure count
                   8.2.2:    Apply local pheromone update with penalty (trust factor)
                   8.2.3:    Retrieve task replica from checkpoint manager
                   8.2.4:    Change resource availability indicator to 0 (temporary
                             suspension)
                   8.2.5:    Resubmit retrieved task to the queue (Step 7)
           8.3:    If task execution is completed
                   8.3.1:    Apply local pheromone update with incentive (trust factor)
                   8.3.2:    Release resource
Step 9:  If all tasks completely executed, terminate the algorithm
```

Figure 4.10. DAFTS algorithm

4.5 Summary

The first contribution in DAFTS algorithm is the dynamic evaporation rate calculation which calculates based on the number of tasks and resources. The calculation will be performed every time new batch of tasks is submitted into the grid system. This ensures that the evaporation rate is properly controlled so that it will not be too quick or too slow which will eventually lead to poor pheromone control.

The second contribution of DAFTS is the enhanced local pheromone update to consider trust factor which is determined by the status of task execution and resource execution history to provide better control of pheromone to the resource which will eventually represent the resource fitness during scheduling process by the Ant. The trust factor in the enhanced local pheromone update is based on identified constant to either increase the pheromone upon successful task execution or reduce the pheromone upon execution failure. The enhanced local pheromone update is called when part of the task is successfully executed, part of the task is failed, and the full length of individual task is completely executed.

The third contribution of DAFTS is the introduction of temporary resource suspension to temporarily prevent resource that failed to execute the task from getting new tasks from the queue. This is essential to allow it to recover at least complete the execution of other parallel execution in the resource. The suspension is controlled based on the resource fitness which means that if the resource is fit, the suspension will be released quicker than the resource that is not fit. This ensures that fit resources, despite failing to execute the task, can continue to be utilized to receive new tasks. The amount of suspension can be controlled by changing the decrement factor (defaulted to 1 per processing cycle) to higher value to quicken the suspension release or lower value to slow down the suspension release.

All the listed contributions are incorporated in DAFTS which provide improved scheduling process by considering the resource fitness and resource availability indicator and enhanced local pheromone update process that considers trust factors and resource execution history. In addition to that, effective fault tolerance techniques are applied which are task resubmission based on checkpoint to eliminate the need to reprocess failed task from the beginning which will eventually reduce the execution time, reduce average makespan, reduce average latency, increase throughput, increase execution success rate and improve load balancing.

CHAPTER FIVE

EXPERIMENTAL RESULT

This chapter presents the experimental results of the DAFTS algorithm compared with other algorithms in terms of execution time, throughput, average latency, average turnaround time, execution success rate and load balancing. Section 5.1 covers the experimental design and followed by the parameter tuning experiments in Section 5.2. The experimental results and analysis are presented in Section 5.3 for two main scenarios which are different rates of failures and different numbers of tasks. Lastly, the summary of the chapter is presented in Section 5.4.

5.1 Experimental Design

Experiments are divided into two parts whereby the first part is used to tune the specific parameters of the proposed algorithm to achieve the most optimal results. The complete parameter tuning experiments cover several scenarios in order to find the optimal constants and parameters in the proposed algorithm is presented in Section 5.2. This includes the validation of static and dynamic evaporation rate, incentive and penalty factor and comparison between suspension and non-suspension.

The second part of experiments covers the thorough comparison with TACO (Wenming et al., 2009), FTACO (Prashar et al., 2014), ACO and ACOwFT (Idris et al., 2017) as presented in Section 5.3. The first scenario is to measure the effectiveness of DAFTS using different failure rates in terms of execution time, throughput, average makespan, average latency, load balancing and execution success rate. The second scenario is to measure the effectiveness of DAFTS using different numbers of tasks in terms of execution time, execution success rate and load balancing. For each scenario, the average of 10 executions is taken as the final results to preserve the consistency and validity of the results.

5.2 DAFTS Parameter Tuning

Before comparison with another algorithm can be performed, it is essential to tune the parameters of DAFTS algorithm so that it can achieve optimal performance. Parameters tuning is also important in adjusting the preference of an algorithm. For instance, it is possible to increase the execution success rate by assigning tasks to fit resources and not utilizing unfit resources, but the drawback will be poor resource utilization or load balancing. It is also possible to reduce overhead by not implementing fault tolerance techniques such as checkpoint and suspension, but this will lead to an increase in execution time.

5.2.1 Dynamic Evaporation Rate versus Fixed Evaporation Rate

Evaporation rate is an important parameter in the pheromone update formula whereby the higher the evaporation rate, the faster the rate of pheromone evaporation. In contrast, a lower evaporation rate results in a slower rate of pheromone evaporation. According to the experiment, a fixed evaporation rate is not effective in controlling the load balancing as it does not consider the number of tasks and resources. A fixed evaporation rate is suitable in a system that does not have faults and has predicted or well-timed tasks submission. This is the reason why the dynamic evaporation rate is proposed which considers the number of tasks and resources to control the rate of pheromone evaporation so that the load balancing, success rate and execution time aspects are improved. In addition to that, in the actual application, tasks may come by batch and resources count may increase or decrease at different timings. Thus, it is important to dynamically reevaluate the current situation and adjust the evaporation rate accordingly to ensure the system can operate at optimum level at any time.

The experiment was conducted using the parameters shown in Table 5.1 in which the number of tasks is changed to measure the effectiveness of dynamic evaporation rate, the failure percentage is set to 50%.

Table 5.1

Simulation parameters for evaporation rate validation

Parameters	**Values**
No. of resources	100
No. of tasks	1000 / 3000 / 5000
PE rating	50 MIPS
Bandwidth	5000 B/S
No. of machine / resource	1
PE per machine	2
Gridlet length	200000 MI
File size	100 + (10-40%)
Output size	250 + (10-50%)

As shown in Figure 5.1, regardless of the number of tasks, the dynamic evaporation rate has a lower load balancing standard deviation which means that the task distribution is done more effectively than the static evaporation rate. The results suggest that by dynamically assigning the evaporation rate based on the number of resources and tasks, the task distribution will be more balanced and eventually lead to better resource utilization.

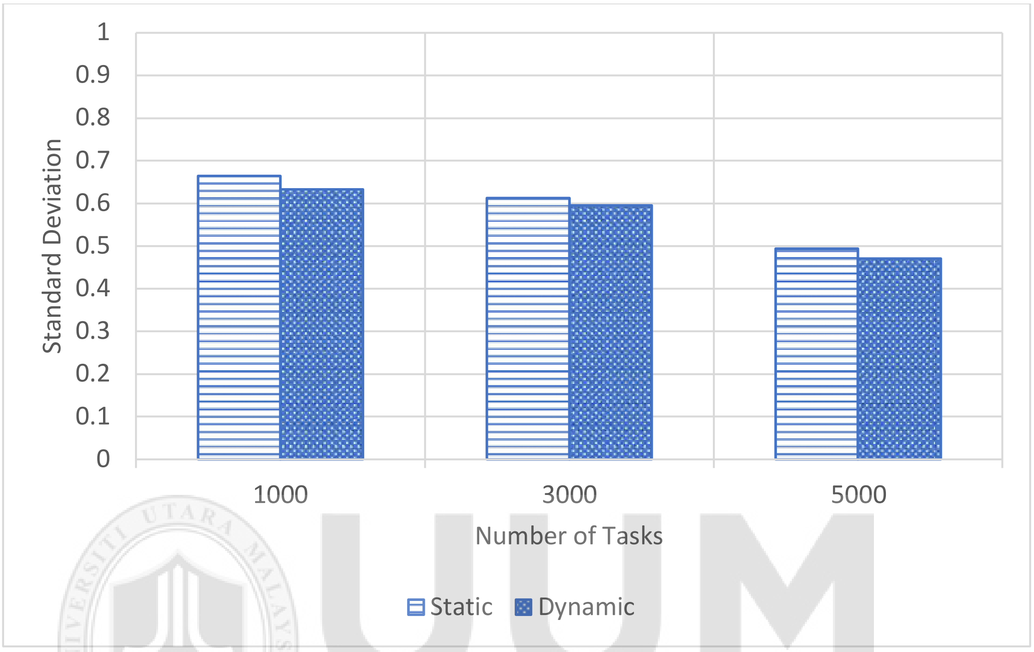


Figure 5.1. Comparison between static and dynamic evaporation rate in terms of load balancing for 100 resources with 1000, 3000 and 5000 tasks

In addition to measuring load balancing, the execution rate should also be considered to complement load balancing. This is because load balancing focuses on task distribution rather than resource execution history. It is still possible that good load balancing can be achieved but with lower execution success rate. The comparison of execution success rate between static and dynamic evaporation is presented in Figure 5.2. In the proposed algorithm, since the resource fitness is considered during task assignment, both load balancing and execution success rate aspects are preserved. For all scenarios, the dynamic evaporation rate produced a higher execution success rate as compared to static evaporation rate.

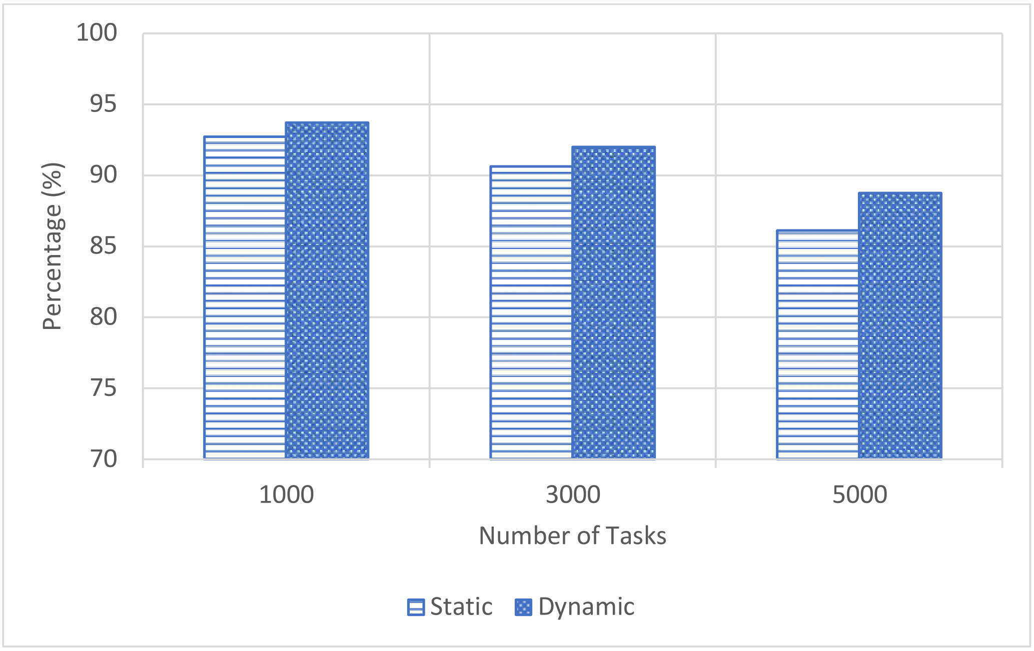


Figure 5.2. Comparison between static and dynamic evaporation rate in terms of execution success rate for 100 resources with 1000, 3000 and 5000 tasks

Last but not least, the execution time, which is influenced by the execution success rate. As depicted in Figure 5.3, in all scenarios, the dynamic evaporation rate resulted in lower execution time as compared to static evaporation rate. The results prove that by complementing between load balancing and execution success rate, the system can achieve a lower execution time. The difference of execution success rate between static and dynamic increases along with the number of tasks incremented. This trend suggests that the larger the ratio of resources and tasks, the better the improvement of execution success rate when the dynamic evaporation rate is applied. There are other aspects that influence the execution time such as latency and average makespan that are indirectly covered by the execution time in this experiment.

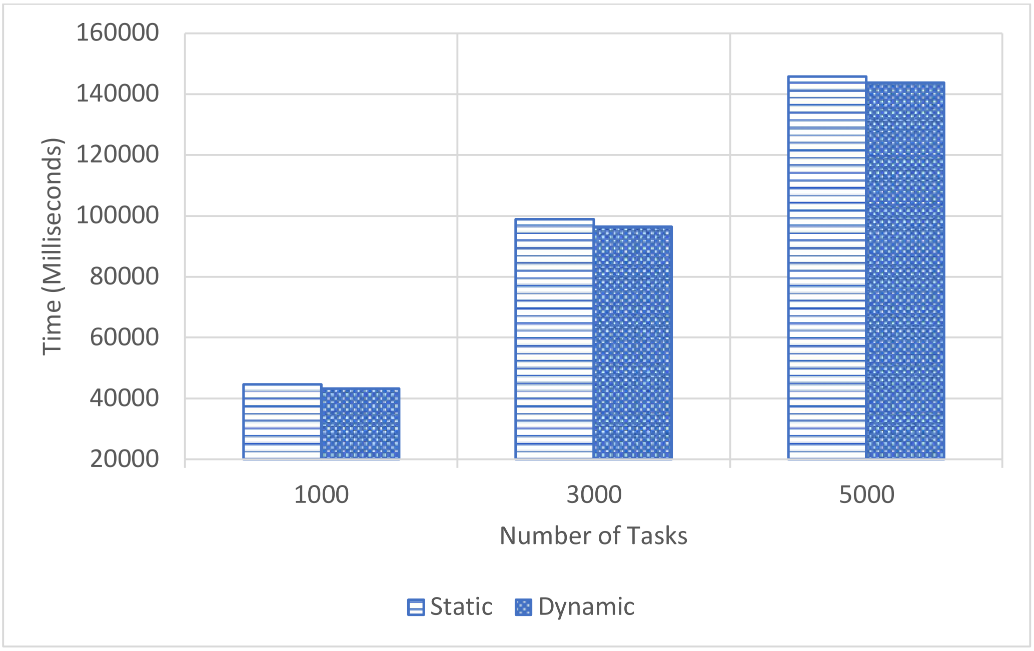


Figure 5.3. Comparison between static and dynamic evaporation rate in terms of execution time for 100 resources with 1000, 3000 and 5000 tasks

Different numbers of tasks require different values of evaporation rates in which large numbers of tasks need slower evaporation rates as compared to small numbers of tasks that require faster evaporation rates. In real situations, there will be different numbers of tasks submitted to the grid broker to the system at different times or schedules. Thus, it is important for the system to have the capability to control the ideal evaporation rate to support execution with effective load balancing control. Adjustment of the evaporation rate can be performed per batch of assigned tasks or at a defined time interval by considering the current tasks and resources available at that time to ensure that the system can operate as optimum level.

5.2.2 Incentive and Penalty Factor

Incentive and penalty or also known as trust factor is proposed to influence the pheromone update process by allowing a more flexible controlling mechanism.

Without an optimal incentive and penalty value, load balancing will be affected as the algorithm would focus on the most fit resources rather than distributing the tasks to all available resources based on their fitness. The optimal values are used to assign variable *T* in (Equation 4.5) so that successful execution will increase the pheromone of a resource, and failure will decrease the pheromone so that the resource will have lesser possibility to be assigned with tasks in following iterations. In this experiment, the optimal values for incentive and penalty are identified based on iterative executions for each combination of incentive ranging from 1 to n and penalty ranging from 1 to 0. To measure the optimal values, the experiment was conducted using the parameters shown in Table 5.2.

Table 5.2

Simulation parameters for incentive and penalty values optimization

Parameters	**Values**
No. of resources	100
No. of tasks	5000
PE rating	50 MIPS
Bandwidth	5000 B/S
No. of machine / resource	1
PE per machine	2
Gridlet length	200000 MI
File size	100 + (10-40%)
Output size	250 + (10-50%)
Incentive Range	[1, 2]
Penalty Range	[0, 1]

A slight difference in the incentive or penalty value provides differences in terms of execution success rate and load balancing. As shown in Table 5.3, the bottom three load balancing values are obtained when the incentive values are 1.3, 1.4 and 1.5 while the penalty value is 1.0.

Table 5.3

Incentive and penalty values optimization for load balancing

		Incentive										
		1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0
Penalty	1.0	0.652	0.077	0.068	**0.048**	**0.047**	**0.047**	0.055	0.063	0.067	0.073	0.097
	0.9	0.713	0.703	0.704	0.682	0.702	0.705	0.707	0.707	0.702	0.699	0.709
	0.8	0.703	0.702	0.703	0.703	0.704	0.711	0.711	0.714	0.718	0.722	0.725
	0.7	0.691	0.686	0.687	0.688	0.690	0.694	0.709	0.714	0.718	0.722	0.727
	0.6	0.704	0.724	0.717	0.681	0.674	0.691	0.710	0.712	0.715	0.717	0.719
	0.5	0.722	0.709	0.709	0.709	0.709	0.728	0.723	0.723	0.723	0.723	0.723
	0.4	0.730	0.698	0.712	0.714	0.716	0.718	0.719	0.721	0.723	0.725	0.727
	0.3	0.702	0.706	0.716	0.717	0.718	0.720	0.721	0.722	0.723	0.725	0.726
	0.2	0.728	0.692	0.694	0.695	0.696	0.710	0.718	0.718	0.719	0.719	0.719
	0.1	0.737	0.748	0.740	0.733	0.726	0.719	0.719	0.720	0.720	0.720	0.722
	0	0.716	0.714	0.715	0.716	0.718	0.727	0.719	0.721	0.722	0.724	0.725

On the other hand, the top three execution success rates are obtained when incentive values are 1.0, 1.4 and 1.5 while the penalty value is 1.0 as listed in Table 5.4. It can be concluded that to achieve the highest execution success rate, the values of incentive and penalty should be 1.0 for both.

Table 5.4

Incentive and penalty values optimization for execution success rate

		Incentive										
		1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0
Penalty	1.0	**94.40**	91.65	92.12	92.43	**92.70**	**93.65**	91.80	91.53	91.27	91.01	90.75
	0.9	82.71	81.09	70.77	72.06	77.22	78.80	77.39	76.65	75.56	73.14	72.72
	0.8	81.93	79.98	67.56	70.78	73.92	76.86	75.12	74.98	74.70	73.14	72.78
	0.7	80.10	78.72	67.00	68.90	70.27	74.87	72.69	69.87	68.57	68.50	67.19
	0.6	79.81	78.26	66.45	67.59	68.26	72.64	72.58	69.12	68.44	68.30	66.96
	0.5	78.51	77.07	65.89	66.23	66.90	71.92	72.26	68.82	68.14	68.00	66.67
	0.4	77.45	76.10	65.33	66.28	66.94	70.70	71.26	67.87	67.20	67.06	65.75
	0.3	76.40	75.12	64.78	64.96	65.61	68.41	68.59	65.33	64.68	64.55	63.29
	0.2	75.35	74.15	64.22	63.66	64.30	65.68	65.53	62.41	61.79	61.66	60.46
	0.1	74.29	73.17	63.66	63.65	64.29	64.25	65.30	62.19	61.58	61.46	60.25
	0	73.24	72.20	63.11	62.34	62.97	63.39	62.52	59.55	58.96	58.84	58.66

This result suggests that when both incentive and penalty are set to 1.0, fit resources are likely to be assigned with the majority of tasks which would lead to stagnation where some resources are heavily loaded. However, the drawback in this situation is that the load balancing of the system will be the worst. Even though the success rate is one of the key criteria in task processing, load balancing is even more important in ensuring that task distribution is undertaken fairly to increase resource utilization. Thus, in the experiments that compare the performance of the proposed DAFTS algorithm with other algorithms, the values of incentive and penalty are set to 1.5 and

1.0 respectively as shown in Table 5.5 where the second highest execution success rate and lowest load balancing standard deviation are achieved.

Table 5.5

Side-by-side comparison of top three figures in Table 5.3 and Table 5.4

Reference Table		Table 5.3			Table 5.4		
		Incentive					
		1.0	1.4	1.5	1.3	1.4	1.5
Penalty	1.0	**94.40**	**92.70**	**93.65**	**0.048**	**0.047**	**0.047**

Incentive and penalty values are assigned as power factors in the pheromone update formula which is very sensitive but effective in manipulating the preference of the algorithm on whether to focus on execution rate but disregard load balancing, or to achieve slightly lower execution success rate but with good load balancing. This value is applied in the local pheromone update formula explained in (Equation 4.5).

The trust factor denoted with T controls the outcome of the calculation. When $T \leq 1$, the calculated value will be reduced and if $T > 1$, the calculated value will be increased. This behavior represents the increase or decrease of pheromone. When T is too small, the decrease of pheromone to the failed resource will be too much and may lead to the resource not getting assigned with task gain. On the other hand, if T is too large, the rapid increase of pheromone may cause fit resources to potentially be assigned with too many tasks, eventually leading to poor load balancing. It is also possible to manipulate the trust factor to achieve specific preference such as to achieve the highest execution success rate without considering the load balancing, or to maximize the resource utilization without the need to achieve highest execution success rate.

5.2.3 Implementation of Suspension Technique

Suspension is proposed to temporarily pause a recently failed resource to allow it to recover and reduce the possibility of another round of failure. The resource suspension considers the initial ratio of tasks to be assigned to each resource and current fitness rate. This experiment is carried out to measure the effectiveness of the suspension as compared to without suspension technique. Execution time, success rate and load balancing are measured to find out whether the proposed suspension technique is effective in optimizing the performance of the DAFTS algorithm. The size of each task is also changed to represent small (50000 MI), medium (200000 MI) and large (1000000 MI). The parameters used in this experiment are presented in Table 5.6. It is hypothesized that the larger the size of tasks, the more effective the checkpointing and resource suspension techniques will be.

Table 5.6

Simulation parameters for resource suspension validation

Parameters	**Values**
No. of resources	100
No. of tasks	5000
PE rating	50 MIPS
Bandwidth	5000 B/S
No. of machine / resource	1

PE per machine	2
Gridlet length	50000 / 200000 / 1000000 MI
File size	100 + (10-40%)
Output size	250 + (10-50%)

Resource suspension is meant to reduce the possibility of another failure should the recently failed resource being assigned with task and allowing the resource to recover in actual implementation. As presented in Figure 5.4, there is a slight reduction to execution time when the suspension technique is enabled with small, medium and large size of tasks. Despite the slight difference in this experiment, when the size is so large in the actual implementation, the difference will be more significant in improving the performance of the system.

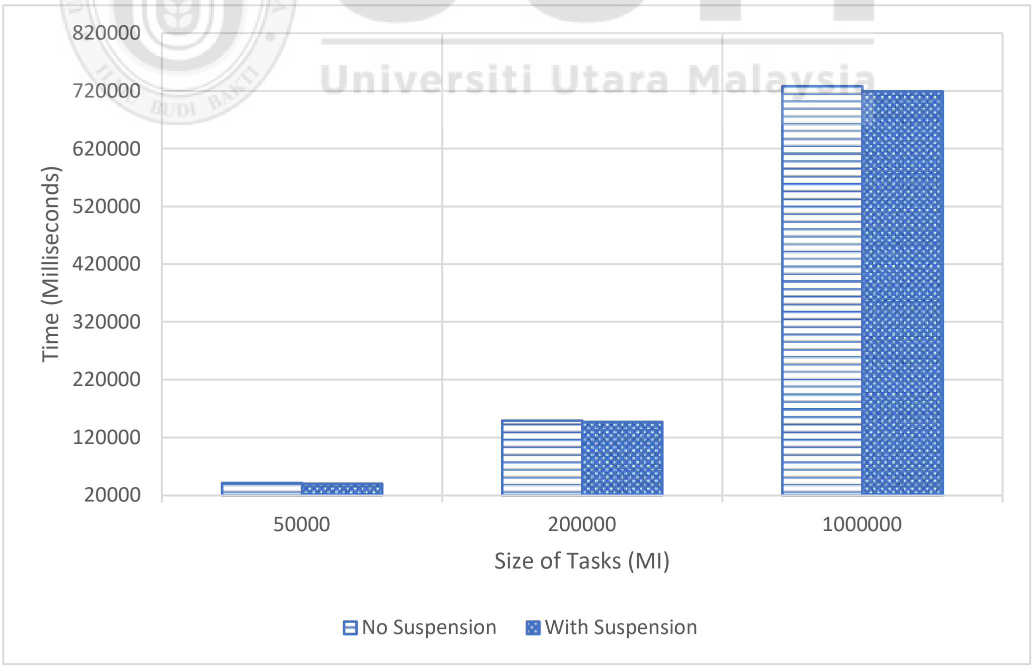


Figure 5.4. Comparison between no suspension and with suspension in terms of execution time for small, medium and large sized tasks

Execution time is directly influenced by the execution success rate as shown in Figure 5.5. The success rate, when the suspension technique is enabled, is higher for all the scenarios as compared to without suspension. It is proven that by temporarily isolating the recently failed resources from being assigned with new tasks, the possibility of failure is also reduced.

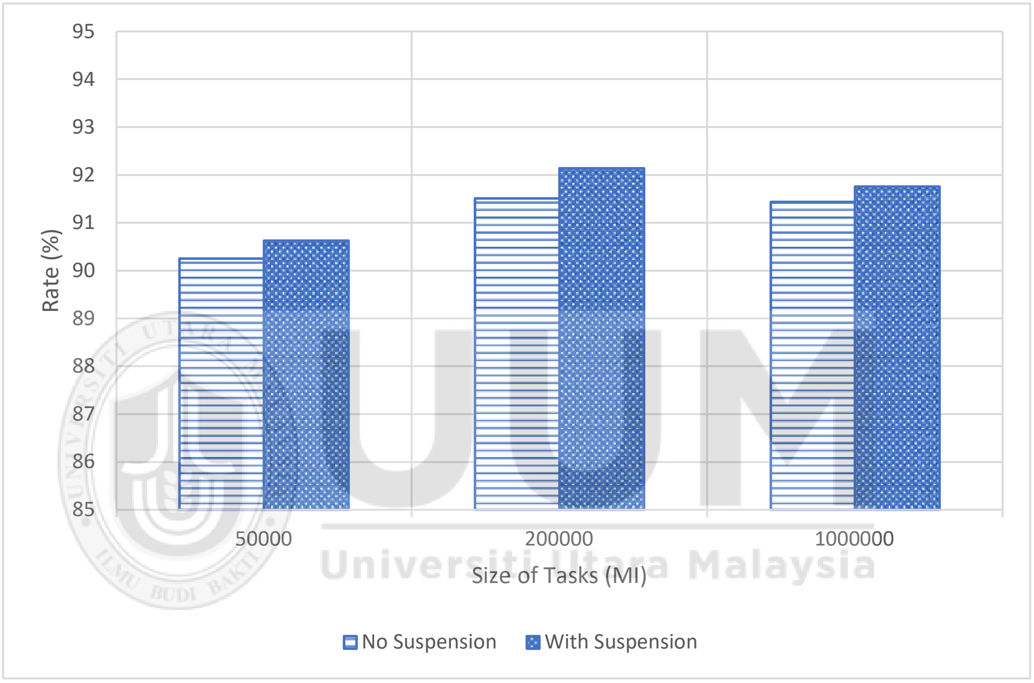


Figure 5.5. Comparison between no suspension and with suspension in terms of execution success rate for small, medium and large sized tasks

In addition to preserving the execution success rate, the load balancing aspect is also considered to ensure that the task distribution is done fairly to avoid bottlenecks. Figure 5.6 depicts that load balancing is further improved when the suspension technique is enabled for all scenarios. The lower the load balancing standard deviation, the better load balancing the system has achieved. This reduction is very significant for small and large sized tasks.

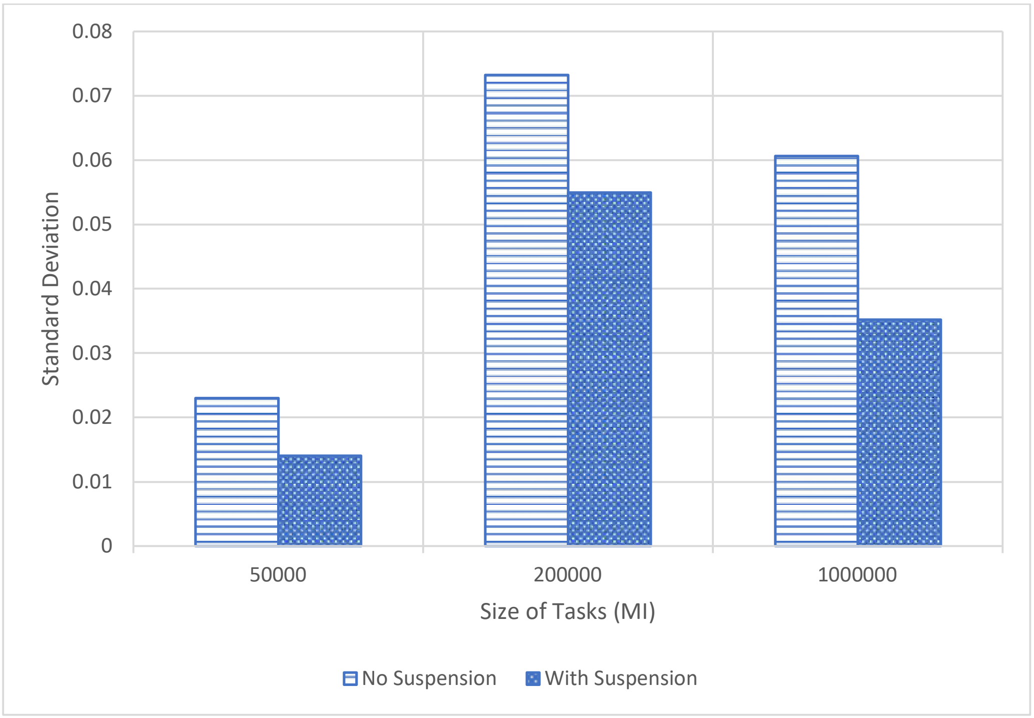


Figure 5.6. Comparison between no suspension and with suspension in terms of load balancing for small, medium and large sized tasks

Temporary suspension provides significant improvement in terms of execution time, execution success rate and load balancing standard deviation. Suspension allows the resource to recover itself by preventing it from getting new load or by user intervention in real applications. In some cases, resources that are constantly overloaded will have higher possibility to fail and by reducing the load may eventually allow it to recover by itself. The length of suspension may vary depending on the number of available resources and the number of tasks to be processed. It is important to set the optimal suspension length so that it will not cause bottlenecks in the task queue which may affect the latency of the system. It is also possible to incorporate a local task queue layered just before the resource so that it can control the flow of queued tasks into each resource during the suspension process to avoid task corruption.

5.3 Results and Analysis

The comprehensive experiments consist of two parts: using different rates of failure between 0% up to 50%; and, using different numbers of tasks ranging between 1000 and 10000. In both parts, all the algorithms are compared thoroughly to validate specific performance metrics. The proposed DAFTS algorithm is compared with TACO (Wenming et al., 2009), FTACO (Prashar et al., 2014), ACO and ACOwFT (Idris et al., 2017) which are re-implemented in GridSim. Each algorithm is executed 10 times for each scenario or interval and the average is taken for a more precise measurement.

5.3.1 Effectiveness of DAFTS to Different Rates of Failure

To validate the performance of the proposed DAFTS algorithm in the presence of failure, a pseudorandom algorithm is used to randomly assign resource fitness within a defined range. In this case, the range of resource fitness is defined between 50% to 100% as used by Amoon (2012) and all other resource and task parameters are adopted from Idris et al. (2017), as shown in Table 5.7, except for resource fitness. For more accurate measurement, each resource is set to have the same PE rating, bandwidth, number of machines and PE per machine.

Table 5.7

Simulation parameters for the effect of different numbers failure rates

Parameters	**Values**
No. of resources	100
No. of tasks	5000

PE rating	50 MIPS
Bandwidth	5000 B/S
No. of machine / resource	1
PE per machine	2
Gridlet length	200000 MI
File size	100 + (10-40%)
Output size	250 + (10-50%)
Resource fitness	50% - 100% (10% interval)

Note. Adapted from Idris et al. (2017).

Execution time is measured from the moment the first task is submitted to the system to undergo scheduling and execution process until all tasks are completely processed. As shown in Figure 5.7, the execution time for DAFTS, ACOwFT and FTACO is incremented gradually as compared to TACO and ACO with rapid increment along a with percentage of fault ranges. This suggests that the checkpoint technique provides significant improvement in terms of execution time as failed tasks do not need to be reprocessed from the initial state. In real implementation, the size of each task is big and requires time to execute. For example, using a non-checkpoint technique, a task that requires one hour to be completely processed may require 1.5 hours to complete if it failed at 50% progress. However, if using the checkpoint technique, the same task may require one hour and several minutes with the assumption that the additional minutes are used to retrieve and reschedule the last saved state into the execution queue.

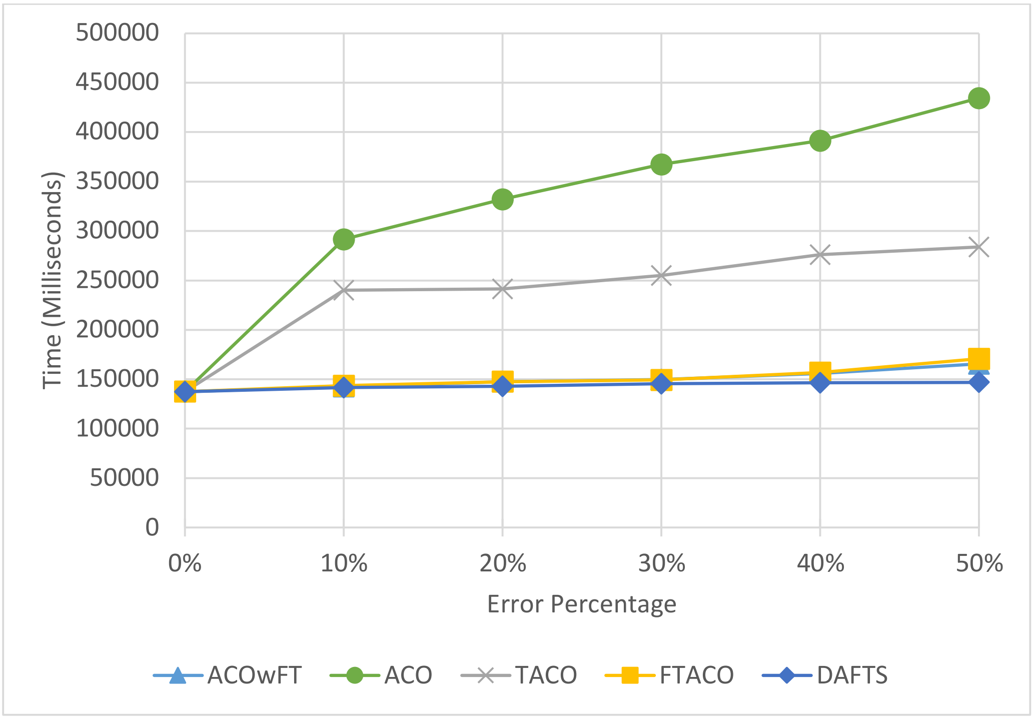


Figure 5.7. Results of execution time for ACOwFT, ACO, TACO, FTACO and DAFTS

For efficiency, throughput is used to measure the performance of the fault tolerance system and calculated by dividing the total number of tasks with total time taken to completely process all tasks. Figure 5.8 shows that DAFTS has the highest throughput while ACOwFT and FTACO have slightly lower throughput. The algorithms with the least throughput are ACO and TACO with more than 50% reduction as compared to the highest throughput algorithms. Since throughput measures the number of tasks completed per unit of time, it is directly influenced by the total execution time. The higher the execution time, the lower the throughput. Thus, one of the ultimate aims of the proposed DAFTS algorithm is to lower the execution time as much as possible in the presence of failure.

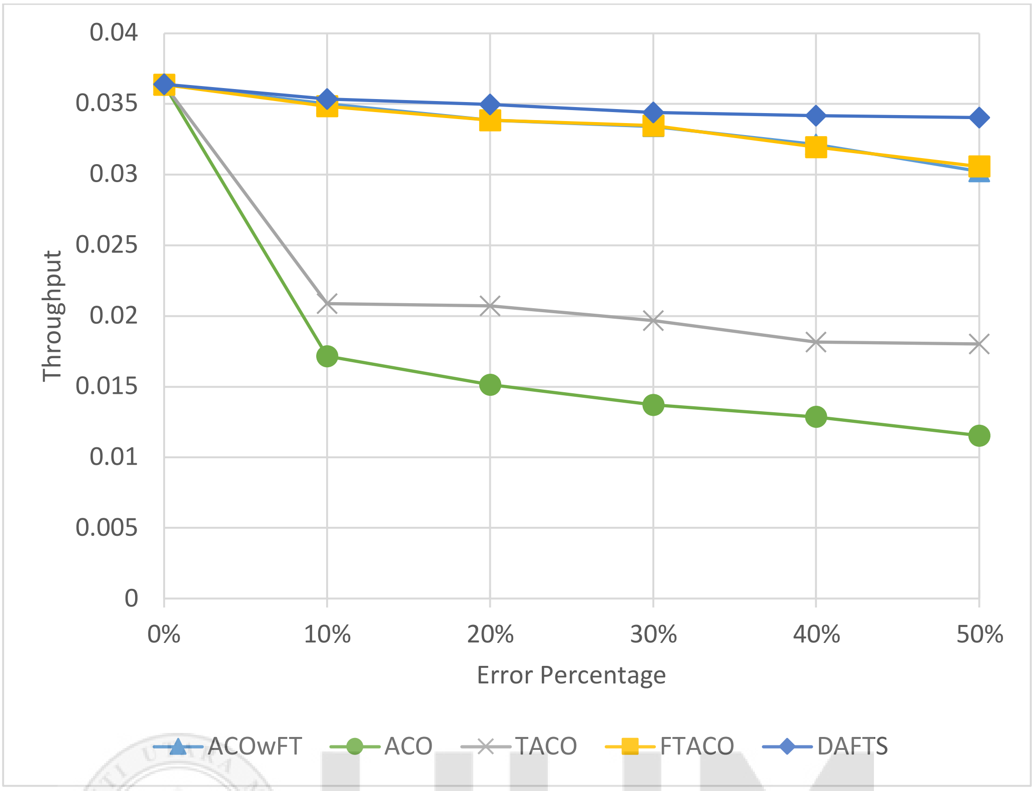


Figure 5.8. Results of throughput for ACOwFT, ACO, TACO, FTACO and DAFTS

Average makespan per gridlet is also considered as average execution time per individual task. In Figure 5.9, ACO has the most average makespan per gridlet followed by TACO with the second highest average makespan. ACOwFT and FTACO have relatively similar performances while DAFTS has the lowest average makespan per gridlet. Average makespan is also related to the total execution time and influenced by the checkpoint technique that allows each failed task to be executed from the last saved state instead of from the beginning. The results also show that consideration of resource execution history with checkpoint technique is effective in reducing the time to process each individual task.

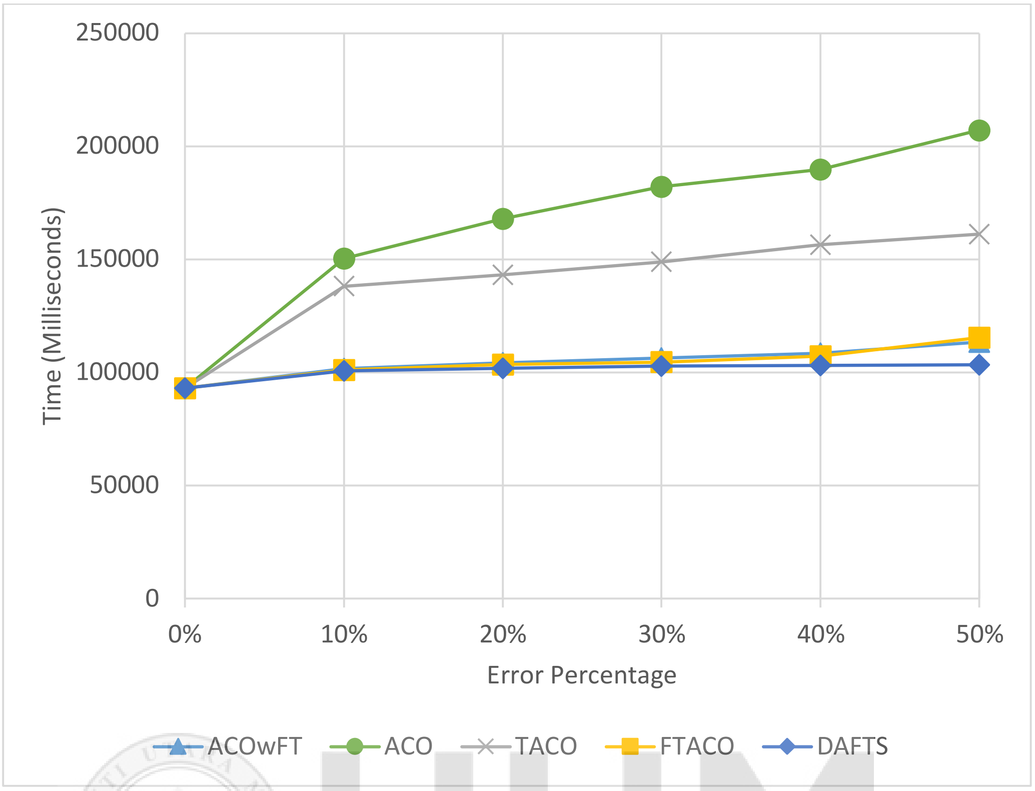


Figure 5.9. Results of average makespan per gridlet for ACOwFT, ACO, TACO, FTACO and DAFTS

Average latency per gridlet measures the waiting time for each gridlet to be processed by an assigned resource. As depicted in Figure 5.10, DAFTS has the lowest average latency followed by ACOwFT and FTACO with second lowest average latency. Both ACO and TACO have the most average latency due to lack of a checkpoint technique that allows a failed task to be reprocessed from the last saved state. The trend of the graph in Figure 5.10 is almost identical to Figure 5.9; this suggests that average makespan and latency have dependency on each other. The results also prove that bias task assignment to only fit resources without proper control can make the waiting queue longer for the resources even though this can improve execution success rate. Thus, it is important to control task assignment based on resource fitness so that the resources with low fitness will still be assigned with the least number of tasks instead of no task at all.

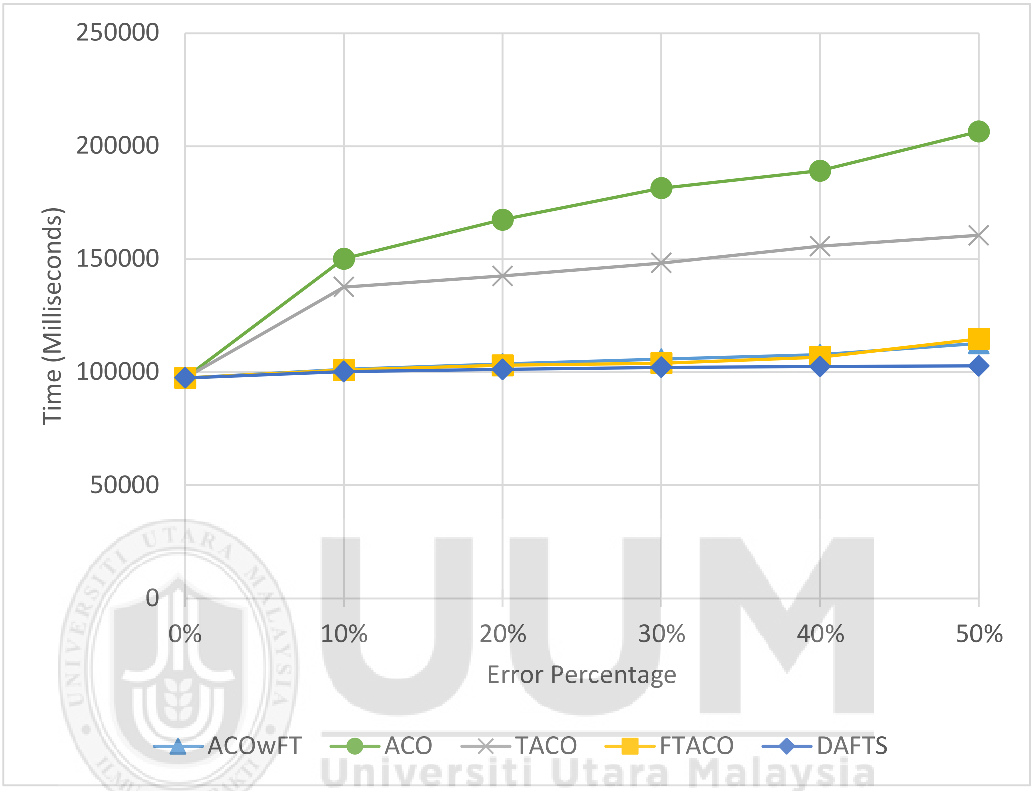


Figure 5.10. Results of average latency per gridlet for ACOwFT, ACO, TACO, FTACO and DAFTS

Load balancing is essential to measure how well the task distribution is performed. This is measured by calculating the standard deviation of initially assigned fitness rate and actual ratio of total processed tasks. As shown in Figure 5.11, ACOwFT has the lowest standard deviation followed by DAFTS with a slightly higher load balancing standard deviation. ACOwFT considers the load of the resource while DAFTS considers the pheromone to balance the task assignment. Even though both FTACO and TACO use pheromone to assign the task, due to a fixed evaporation rate being used, the value of the pheromone cannot be controlled effectively and eventually leads to inconsistency of pheromone level in all resources. In addition, consideration of execution history is effective in determining how fit the resource is in balancing the load. The closer the standard deviation to 0, the better the load balancing. In other words, without even knowing how fit a specific resource is, initially, the proposed algorithm is able to apply heuristic capability to determine the fitness based on execution history while preserving resource utilization.

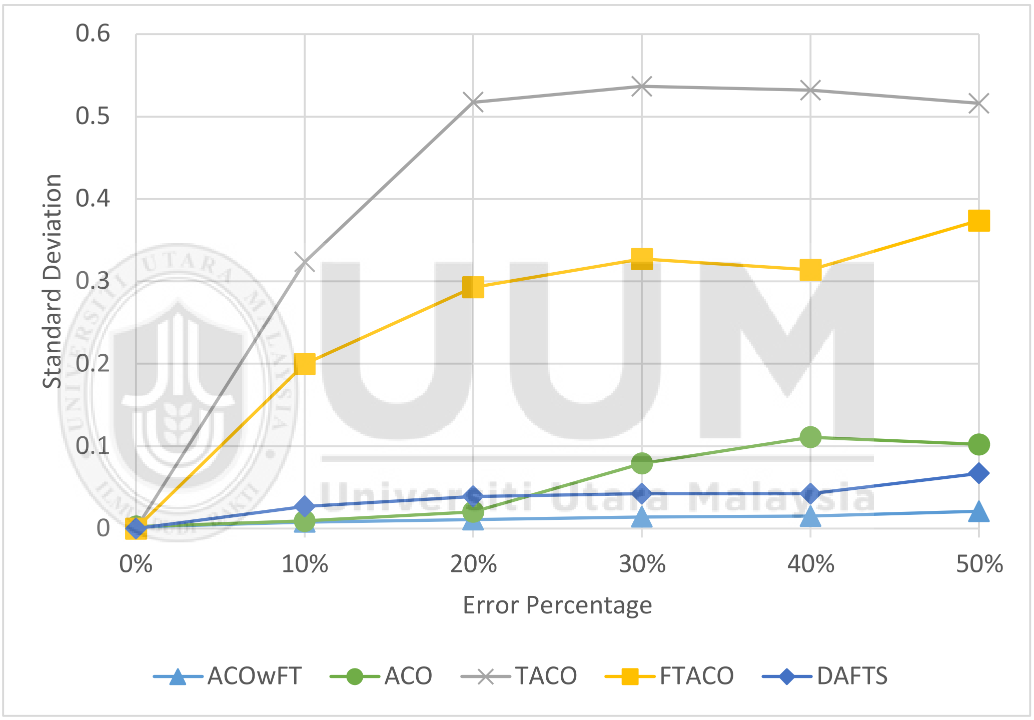


Figure 5.11. Results of load balancing for ACOwFT, ACO, TACO, FTACO and DAFTS

In any fault tolerance system, the ultimate aim is to maintain the execution success rate without disregarding the performance. Figure 5.12 shows that DAFTS has the highest execution success rate followed by TACO. This is because task assignment based on resource fitness reduces the possibility of execution failure. Furthermore, the results prove that task assignment based on resource load is not very effective in reducing the possibility of failure even though it can balance the load effectively. Despite ACOwFT having slightly better load balancing as compared to DAFTS, the proposed algorithm gives a better success rate which is more favorable in the presence of faults.

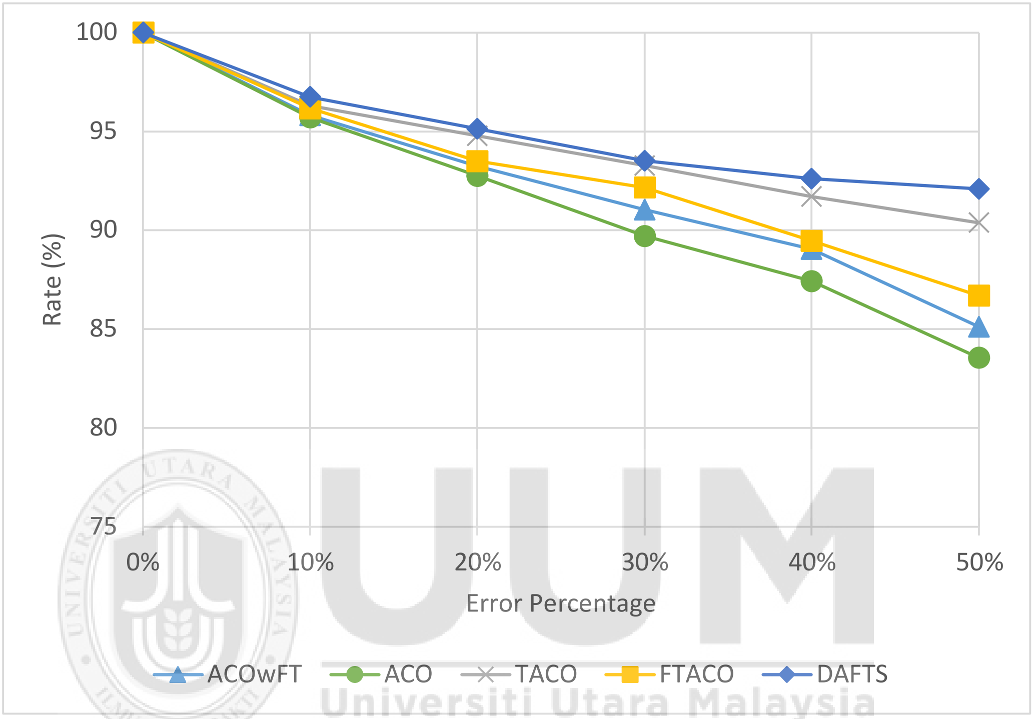


Figure 5.12. Results of success rate for ACOwFT, ACO, TACO, FTACO and DAFTS

Overall, DAFTS produced better performance as compared with the other algorithms especially ACOwFT. In terms load balancing, ACOwFT achieved slightly better performance than DAFTS due to direct consideration of the resource load when assigning tasks. However, in terms of execution time, throughput, latency, makespan and execution success rate, DAFTS outperformed the other algorithms significantly. Thus, it can be concluded that the consideration of resource load would definitely lead to the best load balancing. However, consideration of other factors such as execution history is also favorable in improving the overall performance of the fault tolerance algorithm.

5.3.2 Effectiveness of DAFTS to Different Numbers of Tasks

To further validate the effect of the number of tasks to the performance of all algorithms, an additional experiment is conducted by using different the number of tasks from 1000 to 10000 and resource fitness is distributed between 50% to 100%. The results for each scenario are taken from an average of 10 executions for more consistent results. Table 5.8 summarizes the parameters used where all the parameters except for number of tasks and range of resource fitness are adopted from Idris et al. (2017).

Table 5.8

Simulation parameters for the effect of different numbers of tasks

Parameters	**Values**
No. of resources	100
No. of tasks	1000 – 10000
PE rating	50 MIPS
Bandwidth	5000 B/S
No. of machine / resource	1
PE per machine	2
Gridlet length	200000 MI
File size	100 + (10-40%)
Output size	250 + (10-50%)
Range of resource fitness	50% - 100% (randomized)

Note. Adapted from Idris et al. (2017).

Figure 5.13 shows the effect to the execution time when the number of individual tasks is increased. It can be seen that the execution time increases along with the increase in the number of tasks. However, the increment rate for DAFTS, ACOwFT and FTACO is relatively similar as compared to ACO and TACO that do not employ a checkpoint technique. This result suggests that as the number of tasks increases, the effectiveness of the checkpoint technique, as employed in DAFST, ACOwFT and FTACO, will become more significant.

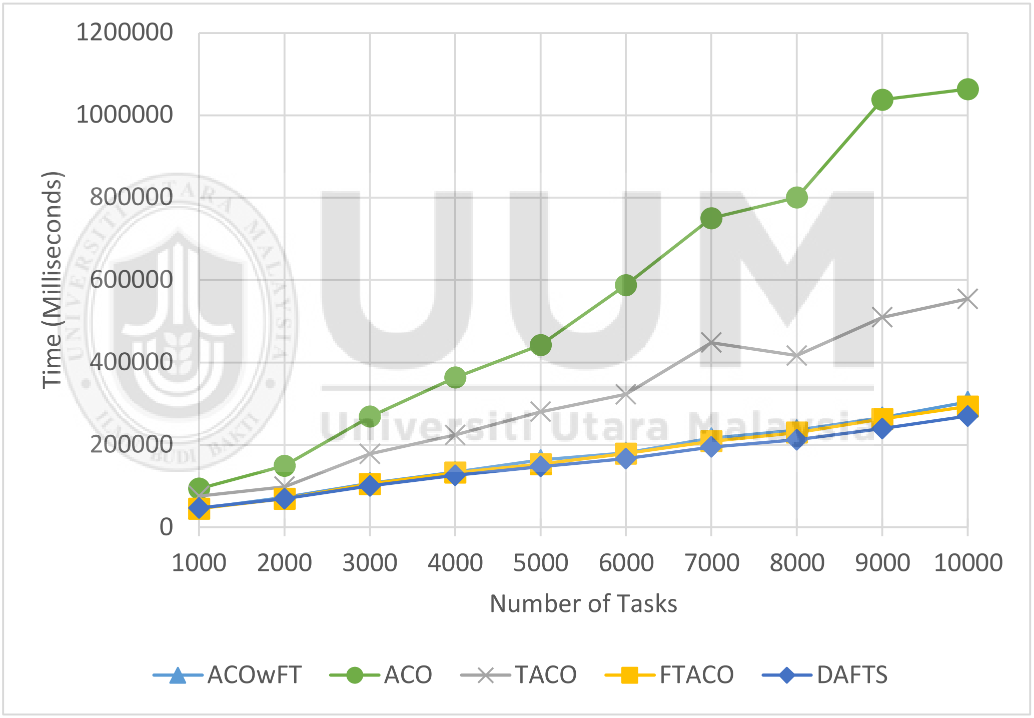


Figure 5.13. Results of execution time for ACOwFT, ACO, TACO, FTACO and DAFTS for different number of tasks

The results of throughput for different number of tasks are shown in Figure 5.14. The throughput is influenced by the number of completed tasks over the execution time. As shown in Figure 5.13, DAFTS achieved the lowest execution time and this is aligned in the throughput results whereby DAFTS outperformed the other algorithms. It is also noted that ant-based algorithms including DAFTS will achieve optimal throughput at certain intervals. This also means that even if the number of tasks is further increased, the throughput will be stagnant since the close to optimal solution is achieved.

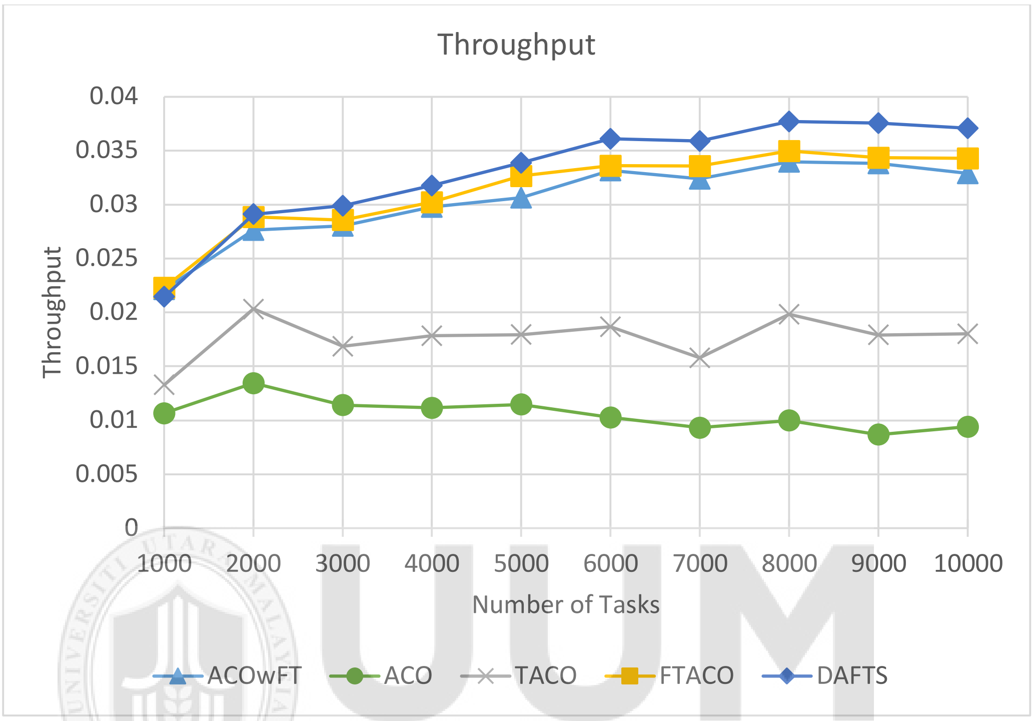


Figure 5.14. Results of throughput for ACOwFT, ACO, TACO, FTACO and DAFTS for different number of tasks

The effects of increasing the number of tasks to average makespan and average latency are depicted in Figure 5.15 and Figure 5.16 respectively. Both performance metrics are directly related to the execution time, thus, the pattern of the graphs are almost identical. It can be seen that DAFTS outperformed the other algorithms in terms of makespan and latency as the task scheduling is done with intention to reduce the number of possible failure by properly assigning the tasks to resources based on fitness rather than load. It can also be noted that the lack of checkpoint technique seems to have major effect to the makespan and latency due to the failed task needs to be reprocessed from the beginning and eventually increases the wait time for the task in queue and task processing time which considers the processing time including when it failed until each individual task is completely processed.

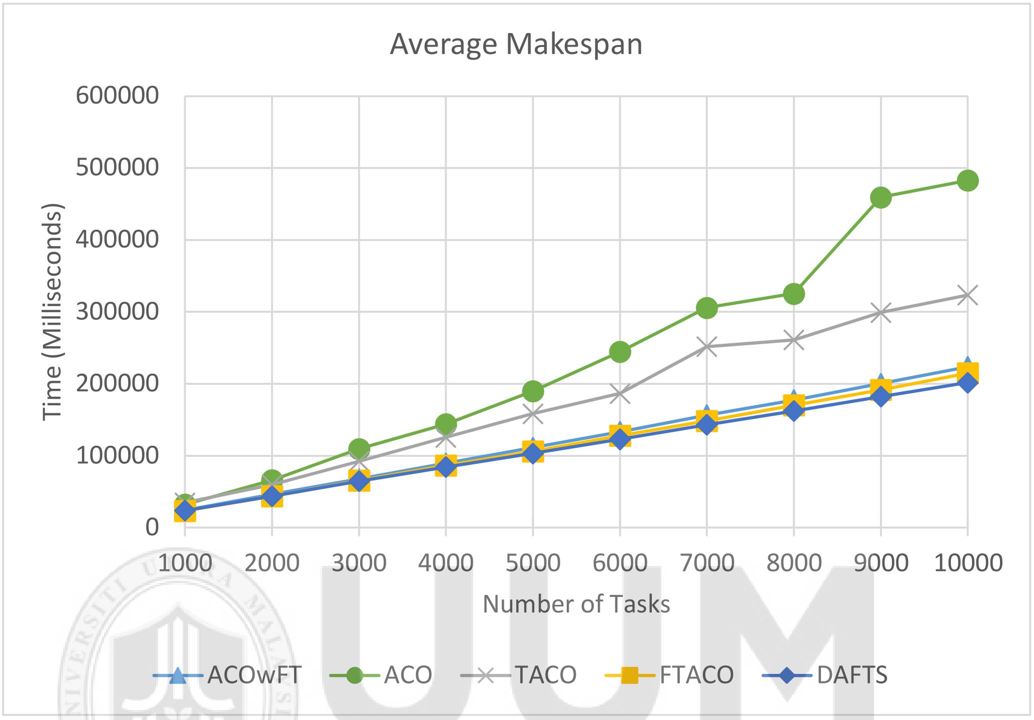


Figure 5.15. Results of average makespan for ACOwFT, ACO, TACO, FTACO and DAFTS for different number of tasks

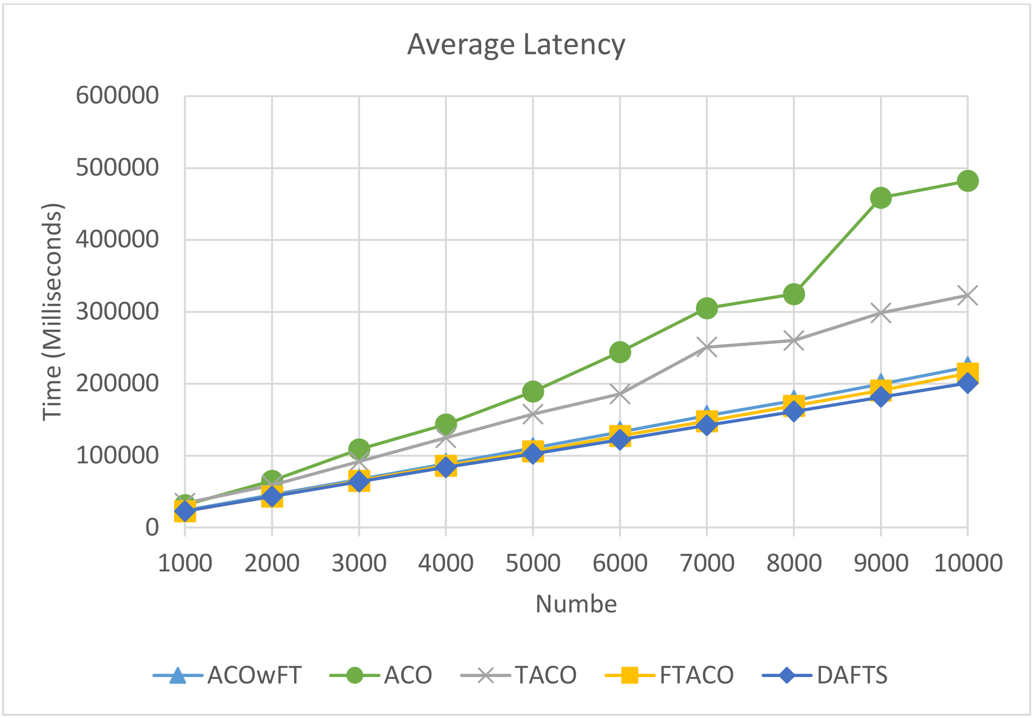


Figure 5.16. Results of average latency for ACOwFT, ACO, TACO, FTACO and DAFTS for different number of tasks

The effect of the number of tasks to the load balancing represented by the standard deviation is shown in Figure 5.17. ACOwFT achieved the lowest load balancing standard deviation followed by DAFTS and ACO. On the other hand, FTACO and TACO have a significantly large load balancing standard deviation. The consideration of resource load or execution history is effective in balancing the task assignment process despite the increase of the number of tasks. Similar to the results in Figure 5.14, load balancing standard deviation will be stagnant at some point and the algorithm is able to adapt with the stability and consistency of the system. This result suggests that in real application, the heuristic information will be carried forward, thus ensuring the subsequent task assignment process happens optimally.

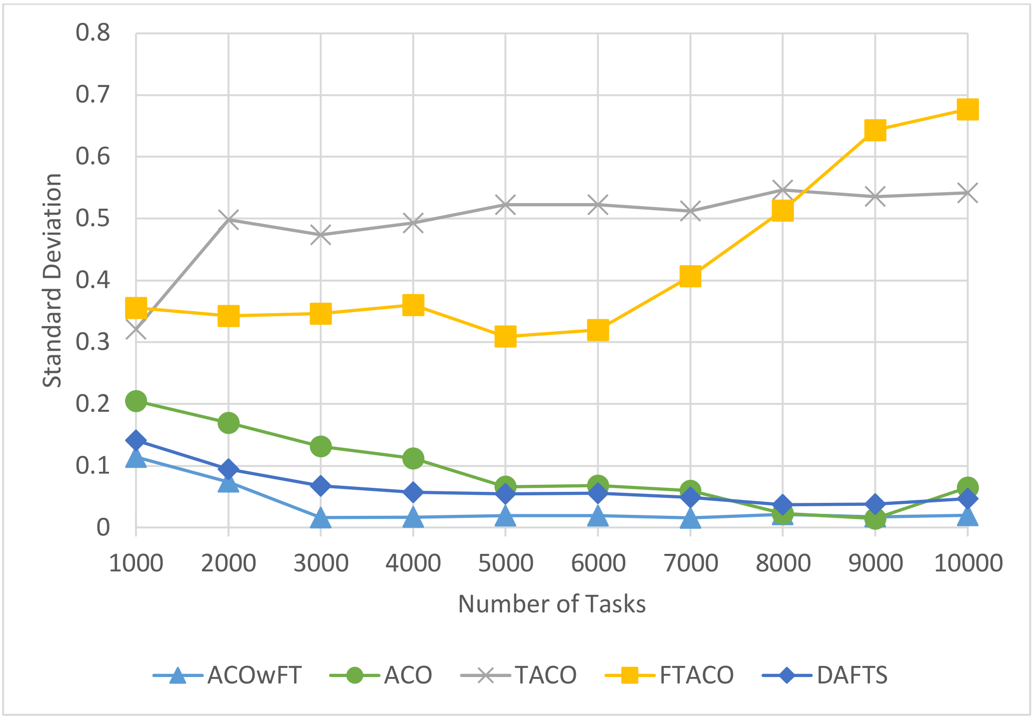


Figure 5.17. Results of load balancing standard deviation for ACOwFT, ACO, TACO, FTACO and DAFTS for different number of tasks

Last, but not the least, is Figure 5.18 which depicts the effect of the number of tasks to the execution success rate. As shown in the results, the execution success rate increases gradually along with the increase in the number of tasks. This behavior is driven by the heuristic information established during the task assignment process in which it will be far from the optimal solution in the beginning but become closer to optimal solution in the later stage. In alignment with the results in Figure 5.14 and Figure 5.17, DAFTS has the highest execution success rate compared to all other algorithms. It is expected that, at some point, the execution success rate will be stagnant for all algorithms regardless of the number of tasks.

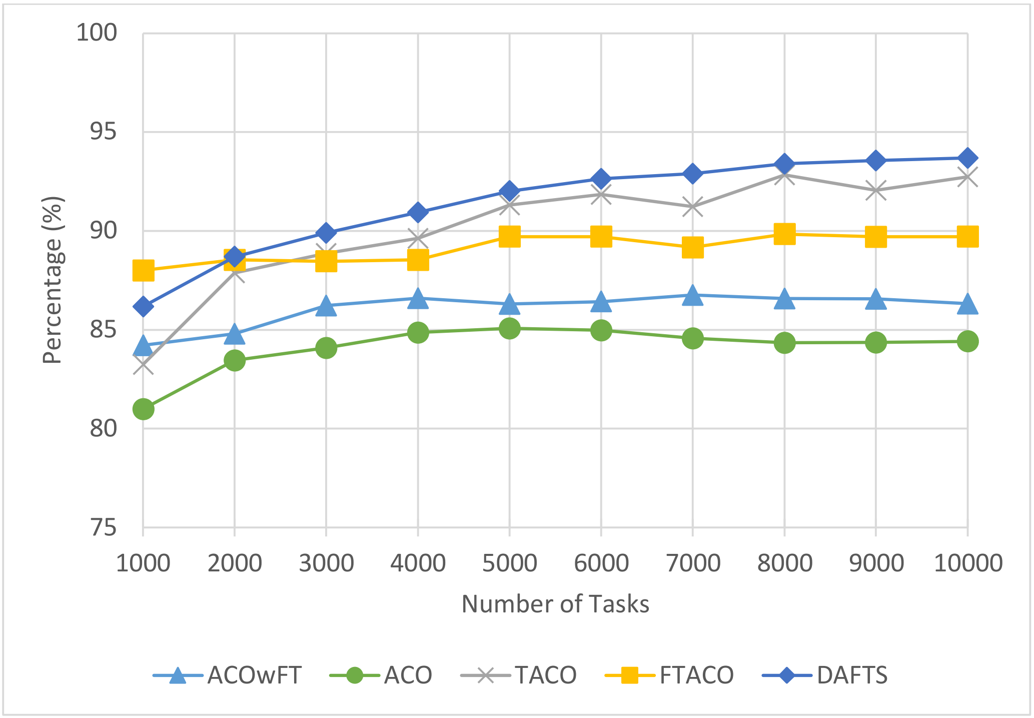


Figure 5.18. Results of execution success rate for ACOwFT, ACO, TACO, FTACO and DAFTS for different number of tasks

An increase in the number of tasks does impact the execution time gradually. However, in terms of execution success rate and load balancing, the increase provides the time needed for the algorithm to achieve the optimal task assignment scheme. Overall, DAFTS achieved the lowest execution time and success rate while ACOwFT with the lowest load balancing standard deviation. Despite the best performance in terms of load balancing, the performance of DAFTS is not that significant as compared to ACOwFT. Thus, it can be concluded that DAFTS achieves the best overall performance due to its execution success rate being significantly higher than ACOwFT.

5.4 Summary

In this chapter, comprehensive experiments were undertaken to determine the specific approaches and optimal parameters to be used by DAFTS to achieve optimal performance. According to the first part of the experiment, implementation of a dynamic evaporation rate and suspension provided slightly better performance as compared to the DAFTS without both approaches. Additionally, the optimal incentive value is 1.5 while the penalty value is 1.0 to achieve the best load balancing with slightly less execution success rate. All these techniques and optimal values are used in the experiments to compare with the other algorithms.

The next experiments were conducted to validate the performance the DAFTS algorithm compared with the other algorithms in terms of execution time, throughput, latency, makespan, execution success rate and load balancing in which the failure rate was changed accordingly. The results suggest that DAFTS achieved the best overall performance despite slightly higher load balancing standard deviation than ACOwFT. However, in terms of execution time, throughput, average makespan, average latency and execution success rate, DAFTS outperformed ACOwFT. Considering the outperformance of DAFTS as compared to slightly higher load balancing standard deviation by ACOwFT, it can be concluded that DAFTS achieved the best overall performance. The results also suggest that all the algorithms that employ the checkpoint technique achieved significantly better performance than those without the checkpoint technique. This outcome is driven by the fact that checkpoint technique eliminates the need to reprocess the failed task from the beginning and eventually lead to lower execution time, lower average makespan, lower average latency, higher throughput and higher execution success rate.

The last experiments were conducted to investigate the effect of tasks' count to the performance of all algorithms in terms of execution time, throughput, average makespan, average latency, load balancing standard deviation and execution success rate. The results suggest that the execution time is directly influenced by the increase of the number of tasks while the throughput, load balancing standard deviation and execution success rate are not directly influenced by the execution time. In fact, the increase in the number of tasks gives more time for the algorithm to achieve the optimal task assignment scheme in which, at some point, the performance will be stagnant despite the increase in the number of tasks. In alignment with the experiment to compare on different rates of failure, DAFTS outperformed the other algorithms in terms of execution time, throughput, average makespan, average latency and execution success rate. In terms of load balancing standard deviation, it achieved slightly higher as compared to ACOwFT that achieved the best. This is influenced by the method used in ACOwFT that considers the resource load during task assignment but in DAFTS, resource fitness and suspension indicator are considered during task assignment process.

Overall, it can be concluded that the proposed DAFTS algorithm has achieved the best performance in terms of execution time, throughput, average makespan, average latency and execution success rate when compared with ACOwFT, ACO, FTACO and TACO. It also achieved insignificantly higher load balancing standard deviation when compared with ACOwFT.

CHAPTER SIX

DISCUSSION

This chapter is dedicated to discuss what have been covered in Chapter 3, Chapter 4 and Chapter 5, and the relationship between all the chapters from the beginning of defining the framework that was used to drive the research process until the getting the results from the designed experiments. Section 6.1 covers relationship between research framework, core design of DAFTS algorithm and associated experiment. Lastly, Section 6.2 summarizes the experimental results.

6.1 Relationship Between Framework, Algorithm Design and Experiment

The research framework in Figure 3.1 in Chapter 3 is designed with phases and methods to align with outcomes that directly related with defined research problems and objectives. As part of fault tolerance techniques identification phase, thorough analysis and review were done on recent works related to fault tolerance in distributed system. The finding is the job reprocessing based on checkpoint and trust factors are the most effective techniques to be applied in DAFTS to ensure that all the failed jobs will be completely processed, the reprocessing is performed from the last saved state instead of from the beginning, and application of trust factors to control the desirability of ants to assign jobs to available resources.

Before the fault tolerance techniques can be incorporated into the DAFTS, the core of the ACS algorithm that focuses on resource assignment and job scheduling is being further enhanced to optimize the performance. The optimization consists of enhancing evaporation rate calculation based on the number of jobs and number of resources, and considering the resource availability indicator. The detailed design of these enhancements is covered in Section 4.2. As depicted in Figure 4.10 in Chapter 4, the dynamic evaporation rate is integrated as soon as the number of jobs and resources are identified in Step 3. Then, the resource availability indicator is being considered to obtain suitable resources as in Step 7.1. Typically, ACO algorithms use fixed evaporation rate which is 0.5 but, in this research, dynamic evaporation rate seems to provide significant improvement over fixed evaporation rate in terms of load balancing, success rate and execution time as presented in Section 5.2.1.

After the improvement of resource assignment and job scheduling process, the improved ACS algorithm is integrated with fault tolerance techniques which are job resubmission to alternative resources based on checkpoint and trust factors that consist of incentive or penalty, and temporary resource suspension. These are presented in Figure 4.10 in Chapter 4, Step 8. The detailed design of temporary resource suspension is explained in Section 4.3. In terms of job resubmission based on checkpoint, the enhanced job scheduling that refers to the resource pheromone will be re-invoked to process remaining checkpoints and jobs. The checkpoint mechanism is also part of the components in the fault tolerance to temporarily store job replicas or checkpoints which will be retrieved back during job reprocessing. As is Section 5.2.2, experiments were done to identify the optimal values for trust factors that consists of 1.5 for incentive and 1.0 for penalty. This constant is used to influence the increase or decrease of resource pheromone based on execution status during local pheromone update process (Step 8.1.2, Step 8.2.2 and Step 8.3.1). In addition to trust factors, resource availability indicator is being toggled to 0 with calculated suspension amount when the job execution has failed (Step 8.2.4). The calculated suspension will evaporate slowly and will toggle back the resource availability indicator to 1 once evaporated completely. The experiments to verify the effectiveness of temporary resource suspension is covered in Section 5.2.3. It can be seen that by applying temporary resource suspension, the DAFTS algorithm achieved improvement in terms of execution time, success rate and load balancing.

The outcome of resource assignment and job scheduling enhancement, and fault tolerance algorithm improvements are finally integrated to form the final DAFTS algorithm. With optimal values for trust factors, and proven techniques such as dynamic evaporation rate and temporary resource suspension, the experiments were carried out to validate the performance against other benchmark algorithms which are TACO (Wenming et al., 2009), FTACO (Prashar et al., 2014), ACO and ACOwFT (Idris et al., 2017) in terms of execution time, success rate, throughput, latency, makespan and load balancing. All the performance metrics used are elaborated in Section 3.4 and Section 3.5. All the benchmarks algorithms were reimplemented in the same simulation environment as DAFTS to ensure fair comparison is performed as presented in Section 5.3.1 and Section 5.3.2.

6.2 Summary of Experimental Result

Two sets of thorough experiments were carried out which are to validate the effectiveness of DAFTS to different rates of failure (Section 5.3.1) and to validate the effectiveness of DAFTS to different numbers of tasks (Section 5.3.2). In the first set of the experiments, the failure rate is being changed within the range of 50% to 100% as it is expected that the higher the possibility of failure, the lower the performance of the algorithm. Table 6.1 shows the summary of performance reduction difference between 0% and 50% failure rate.

Table 6.1

Summary of experiments to validate the performance between 0% and 50% failure rate

Algorithm	**Performance Metrics**					
	Execution Time (%)	Throughput (%)	Average Makespan (%)	Average Latency (%)	Load Balancing Standard Deviation (Difference)	Success Rate (%)
ACO	↑ 216%	↓ 68.3%	↑ 123%	↑ 112%	↑ 0.099	↓ 20%
TACO	↑ 106%	↓ 50.5%	↑ 73%	↑ 65%	↑ 0.516	↓ 11%
FTACO	↑ 24%	↓ 16.0%	↑ 24%	↑ 18%	↑ 0.373	↓ 15%
ACOwFT	↑ 20%	↓ 16.9%	↑ 22%	↑ 16%	↑ **0.019**	↓ 17%
DAFTS	↑ **7%**	↓ **6.49%**	↑ **11%**	↑ **5%**	↑ 0.067	↓ **9%**

As shown in Table 6.1, DAFTS has the lowest percentage change in terms of execution time, throughput, average makespan, average latency and success rate. This indicates that despite the increase of failure rate, the impact of performance is the lowest among all other algorithms as the ants are able to avoid potential failure through consideration of resource fitness. However, ACOwFT has the lowest load balancing standard deviation difference and followed by DAFTS with second lowest. This shows that by considering the resource load or fitness, the load balancing can be preserved. It can also be seen that checkpoint technique implemented in DAFTS, ACOwFT and FTACO significantly reduces the performance degradation on execution time, average makespan and average latency as compared to ACO and TACO that do not implement checkpoint technique. This is because the checkpoint technique allows the failed job to be reprocessed from the last saved state instead of from the initial state. This technique is crucial when the system is dealing with large job size.

The second set of experiments were conducted to measure the effect on performance when the number of jobs is increased while maintaining the same number of resources (100 resources). Table 6.2 summarizes the result of experiments to validate the effect on performance between 1000 jobs and 10000 jobs.

Table 6.2

Summary of experiments to validate the performance between 1000 jobs and 10000 jobs

	Performance Metrics					
Algorithm	Execution Time (Increment Multiplier)	Throughput (Increment %)	Average Makespan (Increment Multiplier)	Average Latency (Increment Multiplier)	Load Balancing Standard Deviation	Success Rate (%)
ACO	↑ 11.3	↓ 11.8%	↑ 15.1	↑ 15.46	0.20 → 0.06	81.0 → 84.4
TACO	↑ 7.35	↑ 36%	↑ 9.3	↑ 9.49	0.32 → 0.54	83.2 → 92.7
FTACO	↑ 6.51	↑ 53.7%	↑ 9.26	↑ 9.39	0.36 → 0.68	88.0 → 89.7

ACOwFT	↑ 6.7	↑ 48.7%	↑ 9.2	↑ 9.43	**<u>0.11 → 0.02</u>**	84.2 → 86.3
DAFTS	↑ **<u>5.8</u>**	↑ **<u>72.9%</u>**	↑ **<u>8.5</u>**	↑ **<u>8.7</u>**	0.14 → 0.05	**<u>86.2 → 93.7</u>**

As shown in Table 6.2, DAFTS has the lowest increment in terms of execution time, average makespan and average latency. In terms of throughput and success rate, DAFTS achieves highest increment. However, in terms of load balancing standard deviation, ACOwFT has the lowest standard deviation and followed by DAFTS. It is clear that when considering the resource load or fitness, the fault tolerance algorithm can improve load balancing due to longer execution time and higher number of iterations that allow ants to produce better scheduling decision. For algorithms that do not consider the resource load such as TACO and FTACO, the load balancing will become unstable as the number of jobs increases. In addition, the application of trust factors as part of pheromone update process in DAFTS and TACO produces the best execution success rate as the unfit resources are punished to reduce the possibility of getting jobs while fit resources are rewarded to increase the possibility of getting more jobs.

In summary, the consideration of resource load leads to the best load balancing and consideration of resource fitness seems to produce good load balancing as well in the presence of faults. Furthermore, trust factors that leads to the highest execution success rate. Application of jobs resubmission based on checkpoint technique resulted to lower execution time, average makespan and average latency.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

DAFTS, as another variant of the ACS-based fault tolerance algorithm, offers the possibility of enhancing the job resubmission process in the presence of faults using task checkpoint and resource suspension techniques. The main aims being to improve the load balancing as well as to increase execution success rate.

Four research questions have been considered and answered by four research objectives corresponding to these questions. The main objective of the research was to develop an improved ACS-based fault tolerance algorithm that can overcome faults by rescheduling a failed task from the last saved checkpoint to another fit resource. To achieve this, the resource fitness is being considered and temporary resource suspension is applied to a recently failed resource to avoid being assigned with another task and in order to undergo the recovery process. In addition to resource fitness, the application of an ACS-based scheduling technique provides better control for the task scheduling process so that it will result in better load balancing in the presence of faults.

The first specific objective was to investigate effective fault tolerance techniques to be applied in DAFTS by considering the objectives of each technique to overcome related problems. The second objective was to improve the ACS-based algorithm to consider the resource fitness during scheduling process, apply temporary suspension to avoid resources that recently failed from getting new tasks, and integrate the trust factors during the pheromone update process. The third objective was to develop the improved algorithm that considers both fault tolerance and load balancing aspects. Last but not the least objective was to evaluate the improved algorithm in simulated grid computing environment by reimplementing benchmark algorithms in the same platform as the DAFTS algorithm.

7.1 Contribution of the Research

The main contribution of this research is the new variant of ACS that provides fault tolerance capability that is based on the way ants search for fit resources to process tasks in a queue, update the pheromone intensity, use of checkpoint technique for task resubmission, search for alternative resources during the task resubmission process, suspend the recently failed resources, and balance the load through fitness-based resource assignment. Within optimal or alternative resources identification, resource execution history is being considered which is represented by the amount of available pheromone. During the pheromone update process, the status of task processing influences the pheromone deposit or evaporation and suspension is defined to temporarily suspend a resource that fails to support task processing. On the other hand, the fault tolerance scheme is coupled with a checkpoint-based resubmission technique to effectively reduce the task reprocessing time should the task fail in the middle of processing.

DAFTS has been proven to effectively reduce execution time, makespan, and latency as well as increase the throughput and execution success rate. The checkpoint-based resubmission technique ensures that the failed task can be reprocessed from the last saved stated instead of from the beginning. Each individual task will be broken down into several checkpoints based on its size. It is important to control the amount of checkpoint calls as an excessive amount may lead to overheads, whereas too few may lead to longer execution times.

DAFTS is equipped with an improved global and local pheromone update which has adopted and adapted the original pheromone update concept from the ACS algorithm. In DAFTS, the global pheromone update is adopted from the original ACS to prevent stagnation. The contribution on this aspect is the enhanced local pheromone update that considers the resource fitness and trust factors to either increase when the resource has successfully executed a task or decrease when the resource fails to execute a task. This action leads to better execution success rate as fit and unfit resources can be easily identified based on the pheromone value.

DAFTS aims to improve load balancing in the presence of faults by using an improved pheromone update technique that considers the dynamic evaporation rate, the resource fitness as well as task processing status when updating the pheromone. Typically, resources that are fit are being over-utilized to preserve the execution success rate of executing tasks. But in DAFTS, it also considers resources that are not fit by assigning small number of tasks and should these small number of tasks fail, standard recovery process will be initiated. Throughout the experiments, DAFTS achieved second best load balancing as compared to ACOwFT that achieved the best load balancing. This is due to the fact that ACOwFT considers the resource load when assigning tasks which directly influences the load balancing. However, the drawback of considering the resource load only is the reduction of execution success rate. The resource load only indicates what is being processed by resources, but not the status of processing. It is possible that resources with low load are not fit which may lead to higher possibility of failure as proven in the experimental results.

Additionally, the task assignment process is improved so that ants consider both the pheromone value and resource availability indicator to find the optimal resources. In typical ant-based scheduling algorithm, pheromone is one of the key criteria in determining which resource to be selected for task assignment. In DAFTS, resource availability indicator is also being considered because recently failed resources will be suspended temporarily, and this indicator will prevent them from being loaded with more tasks. Experimental results showed that this method increases both execution success rate and resource utilization.

Additional contribution includes the proposed formula to measure the load balancing standard deviation for system with faults which is useful in measuring the load balancing of fault tolerance algorithms during experimentations. The proposed formula is meant to measure the deviation of the actual against the expected tasks assignment count to a specific resource. It is designed to be usable in other application domains as well to measure the effectiveness of a fault tolerance algorithm in preserving the load balancing when applying the initial task assignment and reassignment after failure.

Last but not the least is the contribution to the grid computing where the DAFTS algorithm has been designed to work effectively in grid computing to optimize the system in the presence of failures. The contribution is mainly on the new variant of fault tolerance algorithm rather than the architecture of the grid computing.

7.2 Future Work

Grid computing is now being deployed as a subsystem within larger distributed systems such as cloud, cluster and high performance computing which consists of many heterogeneous devices that provide not only processing capability but, also, storage, analytics, artificial intelligence, user interfacing and many more. Future works may include the implementation of DAFTS algorithm in a larger distributed system that is able to provide fault tolerance based on the function of each device and type of failure to improve the efficiency and reliability of the system in performing a required task or function and balance the load.

Another future work could focus on the application of the DAFTS algorithm in other application domains such as the travelling salesman problem, wireless sensor network optimization, timetable and workload scheduling. With the proven results as presented in this research, it is expected that the application of the proposed scheme with minor modifications in other application domains may improve certain aspects such as scheduling, routing and load balancing.

In addition to future works in grid computing and other application domains, potential future work could be on the ACS algorithm to handle simultaneous on-the-fly executions with different characteristics and priorities. This is possible through applying multiple ACS algorithm simultaneously with self-adaptive parameters adjustment on ACO formulae.

REFERENCES

Abdullah, A. M., Ali, A. A., & Haikal, A. Y. (2017). Reliable and efficient hierarchical organization model for computational grid. *Journal of Parallel and Distributed Computing, 104*, 191-205.

Abdullah, A. M., Ali, A. A., & Haikal, A. Y. (2019). A reliable, TOPSIS-based multi-criteria, and hierarchical load balancing method for computational grid. *Cluster Computing, 22*(84), 1-22.

Ahuja, R., & Banga, A. (2019). Resubmission based fault tolerance approach to schedule jobs in grid environment. *EAI Endorsed Transactions on Energy Web and Information Technologies, 6*(24), 1-8.

Alzboon, M. S., Arif, A. S., & Mahmuddin, M. (2016). Towards self-resource discovery and selection models in grid computing. *ARPN Journal of Engineering and Applied Sciences, 11*(10), 6269-6274.

Aliyu, G., Mohammed, A., Abdulmumin, I., Adamu, S., & Jauro, F. (2020). Improving grid computing performance by optimally reducing checkpointing effect. *arXiv, abs/2001.00884.*

Alkhanak, E. N., Lee, S. P., Rezaei, R., Parizi, R. M. (2016). Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. *The Journal of Systems and Software, 113*(2016), 1-26.

Alobaedy, M. M. T. (2015). *Hybrid ant colony system algorithm for static and dynamic job scheduling in grid computing* (Doctoral dissertation). Retrieved from Universiti Utara Malaysia Electronic Theses and Dissertation [eTheses].

Altameem, T. (2013). Fault tolerance techniques in grid computing systems. *International Journal of Computer Science and Information Technologies, 4*(6), 858-862.

Amoon, M. (2012). A fault tolerant scheduling system based on checkpointing for computational grids. *International Journal of Advanced Science and Technology, 48,* 115-124.

Amoon, M. (2013). A job checkpointing system for computational grids. *Central European Journal of Computer Science, 3*(1), 17-26.

Ankita, & Sahana, S. K. (2018). A survey on grid schedulers. In V. Nath, & J. K. Mandal (Eds.), *Nanoeletronics, Circuits and Communication Systems* (pp. 269-279). Singapore: Springer.

Ankita, & Sahana, S. K. (2019). An automated parameter tuning method for ant colony optimization for scheduling jobs in grid environment. *International Journal of Intelligent Systems and Applications, 3*, 11-21.

Arora, R., & Mehta, A. (2018). Resource and task allocation scheduling in distributing system for optimizing execution time, *International Research Journal of Engineering and Technology, 5*(9), 354-360.

Ashraf, I., & Mazher, N. (2013). An approach to implement matchmaking in Condor-G. In *International Conference on Information and Communication Technology Trends* (pp. 200-202). Karachi: FUUAST.

Azeez, I., A., & Haque, S. (2011). Resource management in grid computing: A review. *Greener Journal of Science, Engineering and Technology, 2*(1), 24-31.

Bagherzadeh, J., & MadadyarAdeh, M. (2009). An improved ant algorithm for grid scheduling problem. In *14th International CSI Conference* (pp. 323-328). Tehran: IEEE.

Balasangameshwara, J., & Raju, N. (2012). A hybrid policy for fault tolerant load balancing in grid computing environments. *Journal of Network and Computer Applications, 35*(1), 412-422.

Balpande, M., & Shrawankar, U. (2014). Robust fault tolerance job scheduling approach in grid environment. In *International Conference on Circuits, Systems, Communication and Information Technology Applications* (pp. 259-264). Mumbai: IEEE.

Bansod, R., Virk, R., & Raval, M. (2018). Low latency, high throughput trade surveillance system using in-memory data grid. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems* (pp. 250-253). Hamilton: ACM.

Baru, C., Moore, R., Rajasekar, A., & Wan, M. (1998). The SDSC storage resource broker. In *Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research* (pp. 1-12). Ontario: IBM.

Basu, S. K. (2016). Paradigm and issues. *Parallel and Distributed Computing: Architectures and Algorithms* (pp. 322-352). Delhi: PHI Learning Private Limited.

Bawa, R. K., & Singh, R. K. R. (2012). Application checkpointing in grid environment with improved checkpoint reliability through replication. In *Third International Conference on Computing, Communication and Networking Technologies* (pp. 1-6). Coimbatore: IEEE.

Bienkowski, A. T. (2018). *Resource brokering in grid computing.* (Master's thesis, The University of Western Ontario, Canada).
Retrieved from
https://ir.lib.uwo.ca/cgi/viewcontent.cgi?article=6964&context=etd

Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews, 2*(4), 353-373.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *Journal of ACM Computing Surveys, 35*(3), 268-308.

Bougeret, M., Casanova, H., Robert, Y., Vivien, F., & Zaidouni, D. (2014). Using group replication for resilience on exascale systems. *International Journal of High Performance Computing Applications, 28*(2), 210-224.

Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., & Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing, 61*(6), 810-837.

Brennand, C. A. R. L., Duarte, J. M., Silva, A. P. (2016). SimGrid: A simulator of network monitoring topologies for peer-to-peer based computational grids. In *8th IEEE Latin-American Conference on Communications* (pp. 1-6). Medellin: IEEE.

Bullnheimer, B., Hart, R. F., & Straub, C. (1999). A New Rank-Based Version of the Ant System: A Computational Study. *Central European Journal of Operations Research and Economics, 7*(1), 25-38.

Buyya, R., & Murshed, M. (2002). Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience, 14*, 1175-1220.

Casanova, H. (2001). Simgrid: A toolkit for the simulation of application scheduling. In *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid* (pp. 430-437). Brisbane: IEEE.

Chinnathambi, S., Santhanam, A., Rajarathinam, J., & Senthilkumar, M. (2019). Scheduling and checkpointing optimization algorithm for Byzantine fault tolerance in cloud clusters. *Cluster Computing, 22*(6), 14637-14650.

Chen, X., & Long, D. (2019). Task scheduling og cloud computing using integrated particle swarm algorithm and ant colony algorithm. *Cluster Computing, 22*(2), 2761-2769.

Chowdhury, S., Marufuzzaman, M., Tunc, H., Bian, L., & Bullington, W. (2019). A modified Ant Colony Optimization algorithm to solve a dynamic traveling salesman problem: A case study with drones for wildlife surveillance. *Journal of Computational Design and Engineering, 6*(3), 368-386.

Darmawan, I., & Aradea (2018). Self-adaptive load balancing system for grid computing. *Atlantic Highlights in Engineering, 2*, 43-47.

Dorigo, M. (1992). *Optimization, learning and natural algorithms* (Doctoral dissertation, Politecnico di Milano).

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization.* Cambridge, MA: MIT Press.

Dorigo, M., & Gambardella, L. M. (1997a). Ant colonies for the travelling salesman problem. *Biosystems, 43*(2), 73-81.

Dorigo, M., & Gambardella, L. M. (1997b). Ant colony system: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, *1*(1), 53–66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *The Ant System: An autocatalytic optimizing process* (Technical Report No. 91-016). Milano: Dipartimento di Elettronica, Politecnico di Milano.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant System: Optimization by a Colony of Cooperating Agents. *Journal of IEEE Transactions on Systems, Man, and Cybernetics-Part B, Cybernetics, 26*(1), 29–41

Dill, K. A., & MacCallum, J. L. (2012). The protein-folding problem, 50 years on. *Science, 338*(6110), 1042-1046.

Dumitrescu, C. L., & Foster, I. (2005). GangSim: A simulator for grid scheduling studies. In *IEEE International Symposium on Cluster Computing and the Grid* (pp. 1151-1158). Cardiff: IEEE.

Ebenezer, A. S., Rajsingh, E. B., & Kaliaperumal, B. (2019). A novel proactive health aware fault tolerant (HAFT) scheduler for computational grid based on resource failure data analytics. *International Journal of Computers and Applications, 41*(5), 367-377.

Eleliemy, A., Mohammed, A., & Ciorba, F. M. (2016). Simulating batch and application level scheduling using GridSim and SimGrid. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1-2). Salt Lake City: ACM Digital Library.

El-Zoghdy, S. F., & Alaa, E. (2015). A threshold-based load balancing algorithm for grid computing systems. *Journal of High Speed Networks, 21*(4), 237-257.

Eng, K., Muhammed, A., Mohamed, M. A., & Hasan, S. (2020). A hybrid heuristic of variable neighbourhood descent and great deluge algorithm for efficient task scheduling in grid computing. *European Journal of Operational Research, 1*, 75-86.

Ezzat, A. (2013). *Ant colony optimization approaches for the sequential ordering problem.* (Master's thesis, The American University in Cairo, Egypt). Retrieved from https://pdfs.semanticscholar.org/cdae/d2c3123da91791681255

4ac5f40b67bf85ee.pdf

Fanfakhri, A. B. M., Yousif, A. Y., & Alwan, E. (2017). Multi-objective optimization of grid computing for performance, energy and cost. *Kurdistan Journal of Applied Research, 2*(3), 74-79.

Farid, S., & Hussain, M. (2017). Fault tolerance techniques in cloud and distributed computing: A review. *Technical Journal, 22*(IV), 56-67.

Feng, L., Weiwei, G., & Xiaomin, Z. (2018). Network resource management and scheduling in grid computing. In *2018 International Conference on Robots & Intelligent System* (pp. 207-210). Changsha: IEEE.

Ferdaus, M. H., Murshed, M., Calheiros, R. N., & Buyya, R. (2014). Virtual machine consolidation in cloud data centers using ACO metaheuristic. In F. Silva, I. Dutra, & V. S. Costa (Eds.), *Euro-Par 2014 Parallel Processing* (pp. 306-317). Porto: Springer International Publishing.

Foster, I., & Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications, 11*(2), 115-128.

Foster, I., & Kesselman, C. (2004). The grid in a nutshell. In J. Nabryski, J. M. Schopf, & J. Weglarz (Eds.), *Grid resource management* (pp. 3-13). Boston, MA: Springer US.

Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of High Performance Computing Applications, 15*(3), 200-222.

Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop* (pp. 1-10). Austin, TX: IEEE.

Frey, J., Tannenbaum, T., Livny, M., Foster, I., & Tuecke, S. (2002). Condor-G: A computation management agent for multi-institutional grids. *Journal of Cluster Computing, 5*(3), 237-246.

Gabaldon, E., Guirado, F., Lerida, J. L., Planes, J. (2016). Particle swarm optimization scheduling for energy saving in cluster computing heterogenous environments. In *4th International Conference on Future Internet of Things and Cloud Workshops* (pp. 321-325). Vienna: IEEE.

Gambardella, L. M., Montemanni, R., & Weyland, D. (2012). An enhanced ant colony system for the sequential ordering problem. In *Operations Research Proceedings 2011* (pp. 355-360). Zurich: Springer.

Garba, A., Kana, A. F. D., Abdullahi, M., Abdulmumin, I., Adamu, S., & Jauro, F. (2020). Improving grid computing performance by optimally reducing checkpointing effect. *arXiv preprint arXiv:2001.00884.*

Garg, R., & Singh, A. K. (2011). Fault tolerance in grid computing: State of the art and open issues. *International Journal of Computer Science and Engineering Survey, 2*(1), 88-97.

Garg, R., & Singh, A. K. (2015). Adaptive workflow scheduling in grid computing based on dynamic resource availability. *Engineering Science and Technology, an International Journal, 18*(2), 256-269.

Glaßer, C., Pavan, A., & Travers, S. (2009). The fault tolerance of NP-hard problems. In A. H. Dediu, A. M. Ionescu, & C. Martin-Vide (Eds.), *Language and Automata Theory and Applications* (pp. 374-385). Springer Berlin Heidelberg. doi:10.1007/978-3-642-00982-2_32

Glover, F., & Laguna, M. (2013). Tabu Search∗. In P. M. Pardalos, D. Z. Du, & R. L. Graham (Eds.), *Handbook of Combinatorial Optimization* (pp. 3261-3362). Springer New York.

Goss, S., Aron, S., Deneubourg, J. L., & Pasteels, J. M. (1989). Self-organized shortcuts in the argentine ant. *Naturwissenschaften, 76*, 579-581.

Goswami, S., & Das, A. (2018). Achieving guaranteed service with fault-tolerant resources in grid. Information and Communication Technology, In *Advances in Intelligent Systems and Computing* (pp. 189-196). Springer.

Goyal, S. K., & Singh, M. (2012). Adaptive and dynamic load balancing in grid using ant colony optimization. *International Journal of Engineering and Technology, 4*(4), 167-174.

Grimshaw, A., Ferrari, A., Knabe, F., & Humphrey, M. (1999). Wide area computing: resource sharing on a large scale. *Computer, 32*(5), 29-37.

Guharoy, R., Sur, S., Rakshit, S., Kumar, S., Ahmed, A., Chakborty, S., ... & Srivastava, M. (2017). A theoretical and detail approach on grid computing a review on grid computing applications. In *2017 8th Annual Industrial Automation and Electromechanical Engineering Conference* (pp. 142-146). Jaipur: IEEE.

Gülcü, S., Mahi, M., Ömer, K. B., & Kodaz, H. (2018). A parallel cooperative hybrid method based on ant colony optimization and 3-Opt algorithm for solving traveling salesman problem. *Soft Computing, 22*(5), 1669-1685.

Haider, S., & Nazir, B. (2016). Fault tolerance in computational grids: Perspectives, challenges, and issues. *SpringerPlus 5, 1991*(2016), 1-20.

Haider, S., & Nazir, B. (2017). Dynamic and adaptive fault tolerant scheduling with QoS consideration in computational grid. *IEEE Access*, 5, 7853-7873.

Hanane, H., & Fouzia, B. (2014). Improving resource discovery and query routing in peer-to-peer data sharing systems using gossip style and ACO algorithm. In *The Ninth International Conference on Systems and Networks Communications* (pp. 99-106). Nice: IARIA.

Hajoui, Y., Bouattane, O., Youssfi, M., & Illoussamen, E. (2018). New hybrid task scheduling algorithm with fuzzy logic controller in grid computing. *International Journal of Advanced Computer Science and Applications, 9*(8), 547-554.

Hirofuchi, T., Lebre, A., & Pouilloux, L. (2015). SimGrid VM: Virtual machine support for a simulation framework of distributed systems. *IEEE Transactions on Cloud Computing, 6*(1), 221-234.

Holzinger, A., Plass, M., Holzinger, K., Crisan, G. C., Pintea, C. M., & Palade, V. (2016). Towards interactive machine learning (IML): Applying ant colony algorithms to solve the traveling salesman problem with the human-in-the-loop approach. In *IFIP International Cross Domain Conference and Workshop* (pp. 81-95). Salzburg: Springer.

Hsu, T. S., Wei, H. W., Huang, Y. P., Chen, T. Y., Yeh, T. T., Sun, M. J., Cheng, Y. C., & Shih, W. K. (2014). A digital archive data preservation management system using IRODS architecture. In *International Conference on Computational Science and Computational Intelligence* (pp. 281-284). Nevada: IEEE.

Hwang, K., Dongarra, J., & Fox, G. C. (2012). Grid computing systems and resource management. *Distributed and Cloud Computing: From parallel processing to the internet of things* (pp. 415-473). Waltham, MA: Morgan Kaufmann.

Idris, H., Ezugwu, A. E., Junaidu, S. B., Adewumi, A. O. (2017). An improved ant colony optimization algorithm with fault tolerance for job scheduling in grid computing systems. *PLOS ONE, 12*(5), 1-24.

Imad, E. F., Rachid, S., & El Koutbi, M. (2017). *International Journal of Wireless and Mobile Computing, 12*(2), 154-165.

Ismail, S. A., Ngadi, M., Sharif, J. M., & Kama, M. N. (2017). Authentication mechanisms in a control grid computing environment using identity based identification (IBI). *Advanced Science Letters, 23*(6), 5506-5510.

Jiang, Y., & Chen, W. (2015). Task scheduling for grid computing systems using a genetic algorithm. *The Journal of Supercomputing, 71*, 1357–1377.

Kamra, V., & Chugh, A. (2011). TCP/IP security protocol suite for grid computing architecture. In A. Mantri, S. Nandi, G. Kumar, & S. Kumar (Eds.), *High Performance Architecture and Grid Computing: International Conference on High Performance Architecture and Grid Computing* (pp. 30-35). Chandigarh: Springer.

Kapil, S., Chawla, M., & Ansari, M. D. (2016). On K-means data clustering algorithm with genetic algorithm. In *2016 Fourth International Conference on Parallel, Distributed and Grid Computing* (pp. 202-206). Waknaghat: IEEE.

Karimpour, R., Khayyambashi, M. R., & Movahhedinia, N. (2016). Load balancing in grid computing using ant colony algorithm and max-min technique. *Malaysian Journal of Computer Science, 29*(3), 196-202.

Kaur, P., & Aggarwal, D. (2013). Analysis of fault tolerance on grid computing in real time approach. *International Journal of Scientific & Engineering Research, 4*(11), 817-821.

Kaushik, A., & Vidyarthi, D. P. (2018). A model for resource management in computational grid using sequential auction and bargaining procurement. *Cluster Computing, 21*(3), 1457-1477.

Keerthika, P., & Kasthuri, N. (2011). A new proactive fault tolerant approach for scheduling in computational grid. In *Proceedings on International Conference on Web Services Computing* (pp. 55-59). IJCA.

Keerthika, P., & Kasthuri, N. (2012). An efficient fault tolerant scheduling approach for computational grid. *American Journal of Applied Sciences, 9*(12), 2046-2051.

Keerthika, P., & Kasthuri, N. (2013). An efficient grid scheduling algorithm with fault tolerance and user satisfaction. *Mathematical Problems in Engineering, 2013*, 1-9.

Khaldi, M., Rebbah, M., Meftah, B., & Debakla, M. (2020). Fault tolerance in grid computing by resource clustering. *International Journal of Internet Technology and Secured Transactions, 10*(1-2), 120-142.

Khan, F. (2017). Novel architecture for effective load balancing and dynamic group scheduling in grid computing topology. In *2017 International Conference on Circuits Power and Computing Technologies* (pp. 1-7). Kollam: IEEE.

Khan, S., Nazir, B., Khan, I. A., Shamshirband, S., & Chronopoulos, A. T. (2017). Load balancing in grid computing: Taxonomy, trends and opportunities. *Journal of Network and Computer Applications, 88*(2017), 99-111.

Kim, S., Kim, J., & Weissman, J. B. (2014). A Security-enabled grid system for MINDS distributed data mining. *Journal of Grid Computing, 12*(3), 521-542.

Krasovec, B., & Filipcic, A. (2019). Enhancing the grid with cloud computing. *Journal of Grid Computing, 17*, 119-135.

Kumar, A., & Pathak, H. (2018). Fault tolerant resource management scheme for computational grids. In *International Conference on Intelligent Data Communication Technologies and Internet of Things* (pp. 472-481). Cham: Springer.

Kumar, E. S., & Vengatesan, K. (2019). Trust based resource selection with optimization technique. *Cluster Computing, 22*, 207-213.

Kumar, P., & Kumar, R. (2019). Issues and challenges of load balancing techniques in cloud computing: A survey. *ACM Computing Surveys, 51*(6), 120:1-120:35.

Kuo, R. J., & Zulvia, F. E. (2017). Hybrid genetic ant colony optimization algorithm for capacitated vehicle routing problem with fuzzy demand — A case study on garbage collection system. In *4th International Conference on Industrial Engineering and Applications* (pp. 244-248). Nagoya: IEEE.

Kurochkin, I. I., & Gerk, E. A. (2018). Modeling of task scheduling in desktop grid systems at the initial stage of development. In *Proceedings of the VIII International Conference "Distributed Computing and Grid-technologies in Science and Education"* (pp. 293-297). Dubna: CEUR-WS.

Ku-Mahamud, K. R., & Alobaedy, M. M. (2012). New heuristic function in ant colony system for job scheduling in grid computing. In N. Mastorakis, E. Zaitseva, D. Randjelovic, K. K. F. Yuen, C. G. Carstea, S. Capusneanu, & A. Larion (Eds.), *Mathematical Methods for Information Science and Economics* (pp. 47-52). Montreux: WSEAS.

Ku-Mahamud, K. R., Din, A. M., & Nasir, H. J. A. (2011). Enhancement of ant colony optimization for grid load balancing. *European Journal of Scientific Research, 64*(1), 42-50.

Ku-Mahamud, K. R., & Nasir, H. J. A. (2010). Ant colony optimization for job scheduling in grid computing. In *Fourth Asia Conference on Mathematical/Analytical Modelling and Computer Simulation* (pp. 40-45). Bornea: IEEE.

Kong, X., Shen, H., Chen, X., Wang, C., & Song, C. (2010). Dynamic grid scheduling algorithm based on self-adaptive Tabu Search. In *International Conference on Computer Design and Applications* (Vol. 2, pp. V2-271 - V2-274).

Lai, D. S., Demirag, O. C., & Leung, J. M. (2016). A tabu search heuristic for the heterogeneous vehicle routing problem on a multigraph. *Transportation Research Part E: Logistics and Transportation Review, 86*, 32-52.

Lebre, A., Legrand, A., Suter, F., & Veyre, P. (2015). Adding storage simulation capacities to the simgrid toolkit: Concepts, models, and API. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (pp. 251-260). Shenzhen: IEEE.

Lecca, G., Petitdidier, M., Hluchy, L., Ivanovic, M., Kussul, N., Ray, N., & Thieron, V. (2011). Grid computing technology for hydrological applications. *Journal of Hydrology, 403*(1-2), 186-199.

Levitin, G., Xing, L., Johnson, B. W., & Dai, Y. (2018). Optimization of dynamic spot-checking for collusion tolerance in grid computing. *Future Generation Computer Systems, 86*, 30-38.

Li, F. F., Xie, G., Qi, D. Y., Luo, F., & Xie, D. Q. (2011). Research on novel fuzzy and intelligent resource management in grid computing. In W. Hu (Ed.), *Electronics and Signal Processing* (pp. 1-7). Nanchang: Springer Berlin Heidelberg.

Lin, S. W., & Vincent, F. Y. (2012). A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research, 217*(1), 94-107.

Liu, F., & Guo, W. (2019). Optimized min-min dynamic task scheduling algorithm in grid computing. In *International Conference on Applications and Techniques in Cyber Security and Intelligence* (pp. 745-752). Cham: Springer.

Liu, X., Xia, H., & Chien, A. A. (2004). Validating and scaling the MicroGrid: A scientific instrument for grid dynamics. *Journal of Grid Computing, 2*, 141-161.

Liu, X. F., Zhan, Z. H., Deng. J. D., Li, Y., Gu, T., & Zhang, J. (2018). An energy efficient ant colony system for virtual machine placement in cloud computing, *IEEE Transactions on Evolutionary Computation, 22*(1), 113-128.

Llanes, A., Cecilia, J. M., Sánchez, A., García, J. M., Amos, M., & Ujaldón, M. (2016). Dynamic load balancing on heterogenous clusters for parallel ant colony optimization. *Cluster Computin, 19*(1), 1-11.

Lorpunmanee, S., Sap, M. N., Abdullah, A. H., & Chompoo-inwai, C. (2007). An ant colony optimization for dynamic job scheduling in grid environment. *International Journal of Computer and Information Science and Engineering, 1*(4), 207-214.

Madi, M. K., Yusof, Y., Tahir, H. M., Zaini, K. M., & Hassan, S. (2017). Replica maintenance strategy for data grid. *Journal of Telecommunication, Electronic and Computer Engineering, 9*(1-2), 47-51.

Mahato, D. P., Sandhu, J. K., Singh, N. P., Kaushal, V. (2019). Cuckoo Search-Ant Colony Optimization based scheduling in grid computing. On scheduling transaction in grid computing using cuckoo search-ant colony optimization considering load. *Cluster Computing,* 1-22.

Maipan-uku, J. Y., Konjaang, J. K., & Baba, A. I. (2016). New batch mode scheduling strategy for grid computing system. *International Journal of Engineering and Technology, 8*(2), 1314-1323.

Mandloi, S., & Gupta, H. (2013). Adaptive job scheduling for computational grid based on ant colony optimization with genetic parameter selection. *International Journal of Advanced Computer Research, 3*(9), 66-71.

Martin, E., Cervantes, A., Saez, Y., & Isasi, P. (2020). IACS-HCSP: Improved ant colony optimization for large-scale home care scheduling problems. *Expert Systems with Applications, 142*, 112994.

Mathiyalagan, P., Sivanandam, S. N., & Saranya, K. S. (2013). Hybridization of modified ant colony optimization and intelligent water drops algorithm for job scheduling in computational grid. *ICTACT Journal on Soft Computing, 4*(1), 651-655.

Mavrovouniotis, M., & Yang, S. (2013). Adapting the pheromone evaporation rate in dynamic routing problems. In A. I. Esparcia-Alcázar (Ed.), *Applications of Evolutionary Computation* (pp. 606-615). Berlin: Springer Berlin Heidelberg.

Mavrovouniotis, M., & Yang, S. (2014). Ant colony optimization with self-adaptive evaporation rate in dynamic environments. In *2014 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments* (pp. 1-8). Orlando: IEEE.

Meo, P. D., Messina, F., Domenico, R., Sarné, G. M. L. (2015). Improving grid nodes coalitions by using reputation. *Intelligent Distributed Computing VIII*, 137-146.

Merelli, I. (2019). Infrastructure for high-performance computing: Grids and grid computing. *Encyclopedia of Bioinformatics and Computational Biology, 1*, 230-235.

Moallem, A. (2009). *Using swarm intelligence for distributed job scheduling on the grid*. (Master's thesis, University of Saskatchewan, Canada). Retrieved from http://ecommons.usask.ca/bitstream/handle/10388/etd-04132009-123250/thesis.pdf

Modiri, V., Analoui, M., & Jabbehdari, S. (2011). Fault tolerance in grid using ant colony optimization and directed acyclic graph. *International Journal of Grid Computing and Applications, 2*(1), 14-26.

Mollamotalebi, M., Maghami, R., & Ismail, A. S. (2013). Grid and cloud simulation tools. *International Journal of Networks and Communications, 3*(2), 45-52.

Muthu, T. S., & Kumar, K. R. (2017). Hybrid predictive approach for replica replacement in data grid. In *2017 4th International Conference on Advanced Computing and Communication Systems* (pp. 1-5). Coimbatore: IEEE.

Naik, K. J., Jagan, A., Narayana, N. S. (2015). A novel algorithm for fault tolerant job Scheduling and load balancing in grid computing environment. In 2015 International Conference on Green Computing and Internet of Things (pp. 1113-1118. Noide: IEEE.

Nasir, H. J. A. (2020). *Hybridization of enhanced ant colony system and tabu search algorithm for packet routing in wireless sensor network* (Doctoral dissertation). Retrieved from Universiti Utara Malaysia Electronic Theses and Dissertation [eTheses].

Nasir, H. J. A., & Ku-Mahamud, K. R., & Kamioka, E. (2017). Ant colony optimization approaches in wireless sensor network: Performance evaluation. *Journal of Computer Science, 13*(6), 153-164.

Nassiry, A., & Kardan, A. (2009). Grid learning; computer grids joins to e-learning. *World Academy of Science, Engineering and Technology, 3*, 250-254.

Natrajan, A., Crowley, M., Wilkins-Diehr, N., Humphrey, M. A., Fox, A. D., Grimshaw, A. S., & Brooks, C. L. (2004). Studying protein folding on the grid: Experiences using CHARMM on NPACI resources under Legion. *Concurrency and Computation: Practice and Experience, 16*(4), 385-397.

Nishant, K., Sharma, P., Krishna, V., Gupta, C., Singh, K. P., Nitin, N., & Rastogi, R. (2012). Load balancing of nodes in cloud using ant colony optimization. In *UKSim 14th International Conference on Modelling and Simulation* (pp. 3-8). Cambridge: IEEE.

Obali, M., & Topcu, A. E. (2015). Comparison of cluster, grid and cloud computing using three different approaches. In *2015 23nd Signal Processing and Communications Applications Conference* (pp. 1-4). Malatya: IEEE.

Omer, K., & Abdalla, G. M. T. (2018). Dynamic algorithms replication using grid computing. In 2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (pp. 1-6). Khartoum: IEEE.

Pajorova, E., & Hluchý, L. (2012). Visualization the Natural Disasters Simulations Results Based on Grid and Cloud Computing. In S. J. Miah (Ed.), *Emerging Informatics – Innovative Concepts and Applications* (pp. 85-100). InTech.

Patel, D. K., Tripathy, D., & Tripathy, C. (2016). An improved load-balancing mechanism based on deadline failure recovery on GridSim. *Engineering with Computers, 32*(2), 173-188.

Patel, S. K., & Sharma, A. K. (2018). Optimization of dynamic resource scheduling algorithm in grid computing environment. *International Journal of Computer Sciences and Engineering, 6*(3), 20-26.

Patel, S. K., & Sharma, A. K. (2019). Improved PSO based job scheduling algorithm for resource management in grid computing. *International Journal of Advanced Technology and Engineering Exploration, 6*(54), 152-161.

Patni, J. C., & Aswal, M. S. (2015). Distributed load balancing model for grid computing environment. In *2015 1st International Conference on Next Generation Computing Technologies* (pp. 123-126). Dehradun: IEEE.

Perretto, M., & Lopes, H. S. (2005). Reconstruction of phylogenetic trees using the ant colony optimization paradigm. *Genetics and Molecular Research, 4*(3), 581-589.

Pooranian, Z., Shojafar, M., Abawajy, J., H., & Singhal, M. (2013). GLOA: A new job scheduling algorithm for grid computing. *International Journal of Artificial Intelligence and Interactive Multimedia, 2*(1), 59-64.

Prajapati, H. B., & Shah, V. A. (2015). Analysis perspective views of grid simulation tools. *Journal of Grid Computing, 13*, 177-213.

Prajapati, R., Rathod, D., & Khanna, S. (2017). Comparison of static and dynamic load balancing in grid computing. *International Journal for Technological Research in Engineering, 2*(7), 1337-1340.

Prajapati, H. B., & Shah, V. A. (2015). Analysis perspective views of grid simulation tools. *Journal of Grid Computing, 13*, 177-213.

Prashar, T., Nancy, & Kumar, D. (2014). Fault tolerant ACO using checkpoint in grid computing. *International Journal of Computer Applications, 98*(10), 44-49.

Qureshi, K., Khan, F. G., Manuel, P., & Nazir, B. (2011). A hybrid fault tolerance technique in grid computing system. *Journal of Supercomputing, 56*(1), 106-128.

Qureshi, M. B., Dehnavi, M. M., Min-Allah, N., Qureshi, M. S., Hussain, H., Rentifis, I., Tziritas, N., Loukopoulos, T., Khan, S. U., Xu, C., Zomaya, A. Y. (2014). Survey on grid resource allocation mechanisms. *Journal of Grid Computing, 12*(2), 399-441.

Rajab, H., & Kabalan, K. (2016). A dynamic load balancing algorithm for computational grid using ant colony optimization. *Indian Journal of Science and Technology, 9*(21), 1-7.

Rakheja, D., Kaur, P., & Rkheja, A. (2014). Performance evaluation of resource scheduling and fault tolerance in grid. *International Journal of Computer and Communication System Engineering, 1*(1), 15-19.

Rathore, N. (2015). Efficient agent based priority scheduling and load balancing using fuzzy logic in grid computing. *i-manager's Journal on Computer Science, 3*(3), 7-18.

Rathore, N. (2017). Checkpointing: Fault tolerance mechanism. *i-manager's Journal on Cloud Computing, 4*(1), 28-35.

Rathore, N., & Chana, I. (2014). Load balancing and job migration techniques in grid: A survey of recent trends. *Wireless Personal Communications, 79*(3), 2089-2125.

Rathore, N., & Chana, I. (2015). Variable threshold based hierarchical load balancing technique in grid. *Engineering with Computers, 31*(3), 597-615.

Rubab, S., Hassan, M. F., Mahmood, A. K., & Shah, N. M. (2015). Grid computing in light of resource management systems: A survey. *Journal of Basic and Applied Scientific Research, 5*(5), 33-43.

Sajedi, H., & Rabiee, M. (2014). A metaheuristic algorithm for job scheduling in grid computing. *International Journal of Modern Education and Computer Science, 5*, 52-59.

Santillán, C. G., Reyes, L. C., Conde, E. M., Schaeffer, E., & Valdez, G.C. (2010). A self-adaptive ant colony system for semantic query routing problem in P2P networks, *Computación y Sistemas, 13*(4), 433-448.

Sathish, K., & Reddy, A. R. M. (2017). Workflow scheduling in grid computing environment using a hybrid gaaco approach. *Journal of The Institution of Engineers (India): Series B, 98*(1), 121-128.

Satish, K., & Reddy, A. R. M. (2018). Resource allocation in grid computing environment using genetic–auction based algorithm. *International Journal of Grid and High Performance Computing, 10*(1), 1-15.

Savyanavar, A. S., & Ghorpade, V. R. (2019). Application checkpointing technique for self-healing from failures in mobile grid computing. *International Journal of Grid and High Performance Computing, 11*(2), 50-62.

Schyns, M. (2015). An ant colony system for responsive dynamic vehicle routing, *European Journal of Operational Research, 245*(3), 704-718.

Severance, C. (2014). Ian Foster and the Globus Project. *Computer, 47*(11), 10-11.

Seelwal, P. (2014). Layered mapping of cloud architecture with grid architecture. *International Journal of Computer Science and Mobile Computing, 3*(4), 335-339.

Shah, S. N. M., Mahmood, A. K., Rubab, S., & Hassan, M. F. (2016). Experimental performance analysis of job scheduling algorithms on computational grid using real workload traces. In *1st EAI International Conference on Computer Science and Engineering.* Penang: EAI.

Sharma, P. (2013). Grid computing vs. cloud computing. *International Journal of Information and Computation Technology, 3*(6), 557-582.

Sharma, A., & Bawa, S. (2008). Comparative analysis of resource discovery approaches in grid computing. *Journal of Computers, 3*(5), 60-64.

Sharma, D., & Dalal, S. (2014). Evaluating heuristic based load balancing algorithm through ant colony optimization. *International Journal of Recent Research Aspects, 1*(2), 5-9.

Sharma, D., Sharma, K., & Dalal, S. (2014). Optimized load balancing in grid computing using tentative ant colony algorithm. *International Journal of Recent Research Aspects, 1*(1), 35-39.

Sheikh, S., Shahid, M., & Nagaraju, A. (2017). A novel dynamic task scheduling strategy for computational grid. In *2017 International Conference on Intelligent Communication and Computational Techniques* (pp. 102-107). Jaipur: IEEE.

Sheikh, S., Nagaraju, A., & Shahid, M. (2018). Dynamic load balancing with advanced reservation of resources for computational grid. In P. K. Pattnaik, S. S. Rautaray, & J. Nayak (Eds.), *Progress in Computing, Analytics and Networking,* (pp. 501-510). Singapore: Springer Singapore.

Shukla, Kumar and Singh (2018). An improved resource allocation model for grid computing environment. *International Journal of Intelligent Engineering and Systems, 12*(1), 104-113.

Siegel, J., & Ali, S. (2000). Techniques for mapping tasks to machines in heterogeneous computing systems. *Journal of Systems Architecture, 46*(8), 627-639.

Singh, M. (2016). Incremental checkpoint based failure-aware scheduling algorithm in grid computing. In *2016 International Conference on Computing, Communication and Automation* (pp. 772-778). Noida: IEEE.

Singh, S., & Bawa, R. K. (2016). Proactive fault tolerance algorithm for job scheduling computational grid. *International Journal of Grid and Distributed Computing, 9*(3), 134-144.

Skinderowicz, R. (2017). An improved ant colony system for the sequential ordering problem. *Computers & Operations Research, 86*, 1-17.

Smith, D. J. (2017). *Reliability, maintainability and risk: practical methods for engineers*. Butterworth-Heinemann.

Song, H. J., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., & Chien, A. (2000). The MicroGrid: A scientific tool for modeling computational grids. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing* (pp. 1-22). Dallas: IEEE.

Sotiriadis, S., Bessis, N., Xhafa, F., & Antonopoulos, N. (2012). From meta-computing to interoperable infrastructures: A review of meta-schedulers for HPC, grid and cloud. In *IEEE 26th International Conference on Advanced Information Networking and Applications* (pp. 874-883). Fukuoka: IEEE.

Souli-Jbali, R., Hidri, M. S., & Ayed, R. B. (2019). Impact of replica placement-based clustering on fault tolerance in grid computing. *International Journal of Web Engineering and Technology, 14*(2), 151-177.

Stützle, T & Hoos, H. H. (2000). Max-min ant system. *Future Generation Computer Systems, 16*(8), 889-914.

Sun, W., Zhu, Y., Su, Z., Jiao, D., & Li, M. (2010). A priority-based task scheduling algorithm in grid. In *Third International Symposium on Parallel Architectures, Algorithms and Programming* (pp. 311-315). Dalian: IEEE.

Takefusa, A., Casanova, H., Matsuoka, S., & Berman F. (2001) A study of deadline scheduling for client-server systems on the computational grid. In *Proceedings*

10th IEEE International Symposium on High Performance Distributed Computing (pp. 406-415). San Francisco: IEEE.

Takefusa, A., Matsuoka S., & Nakada, H. (1999). Overview of a performance evaluation system for global computing scheduling algorithm. In *8th IEEE International Symposium on High Performance Distributing Computing* (pp. 97-104). Redondo Beach: IEEE.

Talia, D. (2002). The open grid services architecture: Where the grid meets the web. *IEEE Internet Computing, 6*(6), 67-71.

Tan, W. F., Lee, L. S., Majid, Z. A., & Seow, H. V. (2012). Ant colony optimization for capacitated vehicle routing problem. *Journal of Computer Science, 8*(6), 846-852.

Tiwari, P. K., & Vidyarthi, D. P. (2016). Improved auto control ant colony optimization using lazy ant approach for grid scheduling problem. *Future Generation Computer Systems, 60*(2016), 78-89.

Trasnea, B., Marina, L. A., Vasilcoi, A., Pozna, C. R., & Grigorescu, S. M. (2019). GridSim: A vehicle kinematics engine for deep neuroevolutionary control in autonomous driving. In *Third IEEE International Conference on Robotic Computing* (pp. 443-444). Naples: IEEE.

Vaghela, D. (2014). An advanced approach on load balancing in grid computing. *arXiv preprint arXiv:1409.3651*.

Vansa, R. (2019). *U.S. Patent No. 10,496,618.* Washington, DC: U.S. Patent and Trademark Office.

Venkatesan, R., Ramalakshmi, K., & Latha, D. (2018). Review on Various Approaches for Resource Management and Job Scheduling in Grid Environment. *Journal of Computational and Theoretical Nanoscience, 15*(9-10), 2682-2688.

Vincent, F. Y., Redi, A. P., Hidayat, Y. A., & Wibowo, O. J. (2017). A simulated annealing heuristic for the hybrid vehicle routing problem. *Applied Soft Computing, 53*, 119-132.

Wang, L., Jie, W., & Chen, J. (2018). *Grid computing: infrastructure, service, and applications*. CRC Press.

Wenming, H., Zhenrong, D., & Peizhi, W. (2009). Trust-based ant colony optimization for grid resource scheduling. In *Third International conference on Genetic and Evolutionary Computing* (pp. 288-292). Guilin: IEEE.

Werner, F. (2011). Genetic algorithms for shop scheduling problems: A survey. *Preprint 11*(31), 1-66.

Xhafa, F., Carretero, J., Barolli, L., & Durresi, A. (2007). Requirements for an event-based simulation package grid systems. *Journal of Interconnection Networks, 8*(2), 163-178.

Xia, H., Dail, H., Casanova, H., & Chien, A. A. (2004). The microgrid: Using online simulation to predict application performance in diverse grid network environments. In *Proceedings of the Second International Workshop on Challenges of Large Applications in Distributed Environments* (pp. 52-61). Honolulu: IEEE.

Xu, J., Cai, D., He, J., & Tang, F. (2019). A fault-tolerant routing strategy with graceful performance degradation for fat-tree topology supercomputer. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems* (pp. 405-412). Zhangjiajie: IEEE.

Yadav, K., Jindal, D., & Singh, R. (2013). Job scheduling in grid computing. *International Journal of Computer Applications, 69*(22), 13-16.

Yan, K. Q., Wang, S. S., Wang, S. C., & Chang, C. P. (2009). Towards a hybrid load balancing policy in grid computing system. *Expert Systems with Applications, 36*(10), 12054-12064.

Yang, Z., Ping, S., Aijaz, A., & Aghvami, A. H. (2018). A global optimization-based routing protocol for cognitive-radio-enabled smart grid AMI networks. *IEEE Systems Journal, 12*(1), 1015-1023.

Ye, Z., & Mohamadian, H. (2014). Adaptive clustering based dynamic routing of wireless sensor networks via generalized ant colony optimization. *IERI Procedia, 10*, 2-10.

Younis, M. T., & Yang, S. (2017). Genetic algorithm for independent job scheduling in grid computing. *MENDEL, 23*(1), 65-72.

Younis, M. T., & Yang, S. (2018). Hybrid meta-heuristic algorithms for independent job scheduling in grid computing. *Applied Soft Computing, 72*, 498-517.

Yuan, W., Wang, J., Qiu, F., Chen, C., Kang, C., & Zeng, B. (2016). Robust optimization-based resilient distribution network planning against natural disasters. *IEEE Transactions on Smart Grid, 7*(6), 2817-2826.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing* (Technical Report No. UCB/EECS-2011-82). California: USENIX Association Berkeley.

Zorrilla, M., Flórez, J., Lafuente, A., Martin, A., Montalban, J., Olaizola, I. G., & Tamayo, I. (2017). SaW: Video Analysis in Social Media with Web-Based Mobile Grid Computing. *IEEE Transactions on Mobile Computing, 17*(6), 1442-1455.