Scalable workflow scheduling algorithm for minimizing makespan and failure probability

Maslina Abdul Aziz[1], Izuan Hafez Ninggal[2]
[1]Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA Malaysia, Malaysia
[2]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia

Article Info

Article history:

Received Sep 30, 2018
Revised Oct 18, 2018
Accepted Dec 28, 2018

Keywords:

Makespan
Reliability
Workflow scheduling

ABSTRACT

This paper presents an algorithm called Failure-Aware Workflow Scheduling (FAWS). The proposed algorithm discussed in this paper schedules parallel applications on homogeneous systems without sacrificing the two conflicting objectives: reliability and makespan. The proposed algorithm handles unexpected failure causes rescheduling of the failed task to available resources. In order to analyse the performance of the FAWS algorithm, it will be compared with the popular scheduling algorithm namely Heterogeneous Earliest Finish Time (or HEFT) and Critical Path (CP). A simulation-driven analysis based on realistic workflow application was demonstrated using DAG graph as a continuation of the Layered Workflow Scheduling Algorithm (LWFS). The FAWS algorithm aims to minimize the makespan, increases reliability and therefore boosts the performance of the whole system. A workflow generator was developed to generate large task graphs randomly and scheduled the parallel applications. Based on the simulation results, the proposed algorithm has improved the overall workflow scheduling effectiveness in comparison with existing algorithms.

Corresponding Author:

Maslina Abdul Aziz,
Faculty of Computer and Mathematical Sciences,
Universiti Teknologi MARA Malaysia,
Shah Alam, Malaysia.
Email: maslina@tmsk.uitm.edu.my

1. INTRODUCTION

Many large industrial companies are seeking alternatives to reduce costs, find cheaper ways to develop new services and products as annual costs soar. These companies operate in a dynamic environment where time and cost are the main factors. Moreover, they are at risk to threats and unforeseen incidents. This will because companies re-plan and revise their scheduled plans, cost estimates, risk responses, and others. These problems are unavoidable, but companies need to be able to minimize the risks. Based on research on critical issues and challenges of complex industrial systems, companies can automate their business process and minimize the probability of system failure with the use of workflow scheduling. Among other benefits, it improves efficiency, decision making and process control, and results in better customer service.

One of the main problems with scheduling is finding the optimal schedule. Having an optimal schedule is very challenging because each schedule varies according to its domain and constraints. Since scheduling involves resources, many studies have been carried out on optimizing resource efficiency of workflow schedules involving the homogeneous and heterogeneous issues [1-3]. Makespan can be defined as execution time of a workflow or the length of a schedule. Makespan is also known as deadline. A workflow needs to execute tasks to the assigned processors according to schedule with minimum completion time. Therefore, the main objective is to execute all tasks to the designated resources with-in the given time. Time constraint need to be minimized in the best-effort manner.

Due to the rapid evolution of technology, the need for a large number of complicated applications has increased. Therefore, the Quality of Service (QoS) of service-based systems are becoming increasingly complex [4, 5]. Workflow QoS is composed of different dimensions that are used to characterize workflow schema and instances [6]. Other important QoS parameters highlighted are energy, reliability, and time. Most previous research in workflow scheduling [7-12] are based on limited number of constraints and objectives. Usually, they focused on meeting the deadline or minimizing the makespan. Therefore, they are not adaptable in finding the best trade-offs. Workflows performance can be improved by applying effective scheduling strategies that comprise of calculating and choosing the best task-resource mappings without sacrificing the quality of service objectives.

When executing tasks, workflow application will be scheduled corresponding to the resources available. Thus, the scheduling algorithm or the scheduler will decide which resources will execute the tasks with the goal of minimizing the probability of failure of the application. Due to these issues, this research proposed an algorithm that look for a set of trade-offs between these constraints (reliability and makespan). Large complex processes have high uncertainty that requires multi-objective solutions to solve [9]. A complex application has large number of tasks and therefore it is difficult to schedule. When failure occurs, tasks need to be rescheduled. However, reassigning task is a challenge, since it is important to re-assign the tasks using minimum number of resources [13-15]. Some considerations about rescheduling taks are time requirements and constraints imposed by the service provider for the resource capability [16].

Even though recent studies have suggested and highlighted a number of solutions of improving workflow scheduling performance, however, little attention has been made on to how to optimize the workflow application performance without sacrificing the system reliability. The aim of this workflow scheduling algorithm is to evaluate the performance of the scheduling algorithm that minimizes the total execution time, increases reliability and therefore boosts the performance of the whole system. Therefore, it will improve the workflow scheduling efficiency and the task execution. The objectives are to minimize the makespan and to maximize the reliability of the workflow application. This paper focus on multi-objectives solutions that improves the performance and the reliability of the workflows.

2. RESEARCH METHOD

This section continues with a brief discussion of the new proposed scheduling algorithm called the Failure Aware Workflow Scheduling (FAWS) Algorithm. The comparison of our proposed algorithm is done based on the experiment using DAG graph as a continuation of the earlier algorithm Layered Workflow Scheduling Algorithm (LWFS). This algorithm basic idea is using the execution time of the workflow schedule as the parameter. As shown in the algorithm below, all available processor that is checked on for current backup. The checking process has three main steps. The FAWS algorithm performance is tested based on the simulation using synthetic workflow. The experiment will compare in terms of fault tolerance of the proposed algorithm based on given cases. The task scheduling process will be reassigning the tasks to available resources. However, research proven that the optimization goal is to re-assign the tasks using minimum number of resources [17, 18].

The contribution addressed in this research is to solve reliability workflow scheduling problem. The aim is to design a reliability-aware scheduling algorithm that is capable of workflow execution within ε makespan deterioration. Reliability has become another extremely important issue in workflow scheduling. The reliability-aware scheduling of a workflow application is to maximize the reliability and minimize the makespan of the application.

Given $\mathcal{W} = \{\mathcal{w}_1, \mathcal{w}_1, \dots \mathcal{w}_\mathrm{w}\}$ be set of workflows and $\mathcal{R} = \{\mathrm{r}_1, \mathrm{r}_2, \dots, \mathrm{r}_\mathrm{m}\}$ be a set of homogenous resources such that each $r_1 \in \mathcal{R}$ can experience one or more crash failures at any given time. It is assumed that the communication network is failure-free and a host $r_i \in \mathcal{R}$ can communicate with arbitrary other hosts at the full bandwidth of their network interface.

The challenge addressed on how to schedule the $\mathcal{W}$ workloads on the $\mathcal{R}$ resources such that the $\mathcal{W}$ workflows complete within the deadline and reliability requirements but makes use of minimum number of resources. $\mathcal{W}_{\mathrm{t_i},\mathrm{r_j}}$ Indicating that if task $\mathrm{t_i}$ is assigned to resource $\mathrm{r_j}$. This scheduling problem can be formulated as follows:

$$\max \mathrm{R}_{\mathrm{rely}} \tag{1}$$

$$\mathrm{s.t.}\ \ \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathcal{W}_{\mathcal{w}_\mathrm{i},\mathrm{r_j}} = 1 \qquad \mathcal{w}_1 \in \mathcal{W}, \mathrm{r_j} \in \mathcal{R} \tag{2}$$

$$\sum_{\mathrm{i=1}}^{\mathrm{n}} \mathcal{W}_{(\mathcal{w}_\mathrm{i},\mathrm{r_j})} \le \mathcal{D} \quad and \quad \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathcal{W}_{(\mathrm{t_i},\mathrm{r_j})} \le \mathrm{N}_{\max} \tag{3}$$

$$\mathcal{W}_{w_{i_i},r_j} \in \{0, 1\} \tag{4}$$

In (1) the objective is to maximize the reliability $\left(\mathrm{R}_{\mathrm{rely}}\right)$. The constraint in (2) ensures that each task $\mathrm{t_i} \in \mathcal{T}$ is scheduled on one of the $r_j \in \mathcal{R}$ resources and the constraint in (3) ensures that each $r_j \in \mathcal{R}$ resource executes the tasks without exceeding the deadline $\mathcal{D}$ within the maximum use of resources N.

2.1. Problem Definition

Workflows are often used to represent and model complex distributed scientific computations [1]. A Directed Acyclic Graph (DAG) is the most common abstraction of a workflow. Using a DAG abstraction, a workflow is defined as a $w(\mathcal{T}, \mathcal{L}, \mathcal{D})$ *where* $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ *is a set of tasks represented by* $\mathcal{L} = \{(t_i, t_j, z_{i,j}) \,|\forall\, (t_i, t_j) \in (t_i \; x \; t_j)\}$ *denoting data dependencies between tasks. Each task can have one or more parents or children. The child task cannot start until all parents have finished. In this article, the workflow generator created the workflows with computarion time randomly.*

The overall process of the LWFS is depicted in the following flowchart. Figure 1 presents the flowchart that illustrates the segments and flow of the algorithm. The algorithm is designed to generate a new ranked task list based on parent-child task dependency. The algorithm consists of three parts. The first part is getting input form the user. The user will need to specify number of tasks and layers. When the user enters the number of tasks and layers, a set of tasks and dependencies will be created. The task parent-child dependency for each task in different layers will created. The second part is to rank the tasks based on number of dependencies. The tasks will be automatically distributed between the layers. The ranking of tasks is also based on layers of the workflow. The first node will always be the start task ($\mathrm{t_{entry}}$). The start task will be in Layer 1. Then this process will continue until all tasks are assigned in respective layers. The last task ($\mathrm{t_{exit}}$) will be in the last layer. In the second phase, the 2-ways scanning and mapping of tasks is done. This process allows to capture the exact number of tasks dependencies for each task. The number of dependencies will be totalled up. Then, a new task list will be generated and ready to be scheduled. This new task list contains the new ranked tasks.

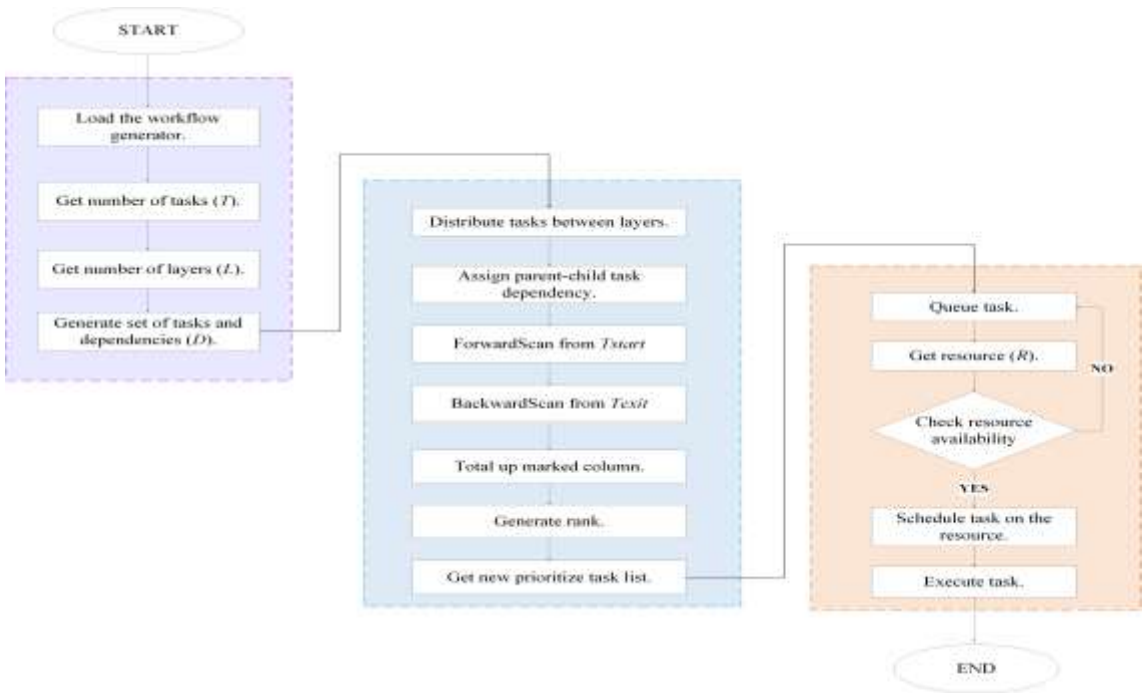
START
Load the workflow generator.
Get number of tasks (T).
Get number of layers (L).
Generate set of tasks and dependencies (D).
Distribute tasks between layers.
Assign parent-child task dependency.
ForwardScan from Tstart
BackwardScan from Texit
Total up marked column.
Generate rank.
Get new prioritize task list.
Queue task.
Get resource (R).
NO
Check resource availability
YES
Schedule task on the resource.
Execute task.
END

Figure 1. Flowchart of FAWS algorithm

The third part is the assignment of task to the resources. The assignment of task depends on the availability of the resources. The availability of resources will be checked before scheduling. If the resource is not available, the task will wait for the next available resource. The tasks are equally mapped to the resources. The task scheduled on the same resource will not incur any communication cost. Therefore, if the

T_1 is scheduled on the R_1, T_2will immediately continue at R_1 also. From the algorithm, a workflow generator is used to generate different workloads.

3. SIMULATION

3.1. Simulation 1

For the simulation studies, random tasks graph with $\mathcal{T} = \{\mathrm{t}_{20}, \mathrm{t}_{40}, \mathrm{t}_{60}, \mathrm{t}_{80} \text{ and } \mathrm{t}_{100}\}$ tasks were generated on a heterogeneous computing system with 5 resources $\mathcal{R} = \{r_1, r_2, r_3, r_4, r_5\}$. The $\mathcal{R}$ processors are assumed to be fault-free and each processor $r_1 \in \mathcal{R}$ can only execute at most one task at a time. Each resource will follow the First In First Out (FIFO) queue that hold the tasks that are scheduled on the resource. The resources will execute each task by task ordering queue without any pre-emption. This simulation focus on handling failures in workflow scheduling. In order to apply the real failure situation, all the tasks generated for this simulation are randomly generated based on computation time for each task and the layers for the DAGs. The execution times of each task on the DAG graphs is assumed to be uniformly distributed between 5 and 20 minutes. The layers were also automatically generated based on the number of tasks entered.

The task dependencies were randomly generated by the workflow generator. The new ranked list that are scheduled to be executed on 5 resources $\mathcal{R} = \{r_1, r_2, r_3, r_4, r_5\}$. This is a two-matrix table that calculate tasks dependencies. A new rank list will be generated.After generating the new list, these tasks will be queued for execution. The tasks are scheduled to resources subject to resource availability. The tasks will start executing based on the task's dependency and for this simulation it depends on layers. The tasks will be distributed to the resources $\mathcal{R}_5 = \{r_1, r_2, r_3, r_4, r_5\}$. Among the matrix combinations of task distribution for 20, 40, 60, 80 and 100 with 5 resources are be shown below.

$$\mathrm{T}_{20} = \begin{array}{l} \mathcal{R}_1 = 4 \\ \mathcal{R}_2 = 4 \\ \mathcal{R}_3 = 4 \\ \mathcal{R}_4 = 4 \\ \mathcal{R}_5 = 4 \end{array} \quad \mathrm{T}_{40} = \begin{array}{l} \mathcal{R}_1 = 9 \\ \mathcal{R}_2 = 9 \\ \mathcal{R}_3 = 7 \\ \mathcal{R}_4 = 8 \\ \mathcal{R}_5 = 7 \end{array} \quad \mathrm{T}_{60} = \begin{array}{l} \mathcal{R}_1 = 13 \\ \mathcal{R}_2 = 13 \\ \mathcal{R}_3 = 11 \\ \mathcal{R}_4 = 11 \\ \mathcal{R}_5 = 12 \end{array} \quad \mathrm{T}_{80} = \begin{array}{l} \mathcal{R}_1 = 16 \\ \mathcal{R}_2 = 15 \\ \mathcal{R}_3 = 17 \\ \mathcal{R}_4 = 17 \\ \mathcal{R}_5 = 15 \end{array} \quad \mathrm{T}_{100} = \begin{array}{l} \mathcal{R}_1 = 21 \\ \mathcal{R}_2 = 20 \\ \mathcal{R}_3 = 22 \\ \mathcal{R}_4 = 18 \\ \mathcal{R}_5 = 19 \end{array}$$

For this simulation, a parallel *R* resources, where each resource is a processing element is subject to its own individual failures. When a resource suddenly stops functioning, the queued tasks need to be rescheduled; How to reschedule these tasks in *n-1* given resources? When the resource R_3 failed on the 20th minute, task T_3 is in the queue and almost finishing. Due to the resource failure, earliest available time is checked for the remaining resources. Since all resources are occupied, T_3 has to wait until the next available resource R_2, after the completion of T_{11} at 21 minutes. Since T_{11} and T_3 are in the same layer two in the DAG graph, these tasks have the same priority. Task T_3 will restart at time 22nd minute. Since T_3 needs another 10 minutes to complete, this will extend the whole queue. When a failure happens, the algorithms need to optimize both makespan and reliability.

The simulation applies checkpointing mechanism with restart mechanism. The pointers are fixed at time 20, 40, 60 and 80 minutes. If the pointer detects resource failure, the task will restart immediately. The failed task will be assigned to the next available resource. Since these tasks are dependent with each other, the task at the same layer can be scheduled freely as these tasks have same rank. However, as proven in the previous chapter, the task that has high number of children need to be scheduled first. This is because these are critical tasks that will delay the overall makespan.

3.2. Simulation 2

For simulation 2, the situation happens when one of the resources fails in the middle of task execution and restart on current node. As compared to simulation 1, the resource failed from the middle till the end of the task execution. The result shows the response of resources to resource failure with respect to Condition 2. As it can be seen in this graph, all tasks are in the queue for execution are affected. Generally, if the resource stopped working, the incomplete task will be rescheduled and restart from the beginning. In this simulation, there are 100 tasks to be executed among 5 resources. The task distribution matrix is shown below. The task distribution matrix showed that R_3 is the high dependent resource since there are 22 tasks queuing to be executed at R_3. For the example below, the workflow application with 100 tasks has 12 layers. T_3 is at layer 2. Layer 2 has 10 tasks. When resource R_3 suddenly failed when executing tasks, the rescheduling of tasks will depend on the next available resources (FIFO) and task ranking (layer). The task needs to be scheduled within the same layer. The proposed algorithm needs to ensure that tasks are executed successfully even when resource failure has occurred and restart in the specified period.

4. RESULTS AND ANALYSIS
4.1. Simulation 1
For this simulation, the check-pointing method are used where there will be fault-tolerant state in case of failure. When a resource fails, the checkpointing method will restart the execution process at the point of failure. The main problem us to minimize the makespan by scheduling the queued tasks in the workflow. One of the disadvantages of checkpointing method is it will incur extra cost. The incur cost depends on the number of pointers used. If the checkpointers are overly used, more cost will incur. If the checkpointers are insufficient, the tasks affect the overall makespan and the performance of the workflow. In this simulation, the workflow complexity of the simple case were tested in 2 conditions; with and without checkpointers. In this simulation, it compares the checkpointing method for different numbers of tasks, specifically focusing on impact on the makespan. Generally, the makespan increases for both scenarios (with and without checkpointing). Based on the result in Figure 2, it shows that the makespan of workflow application increased 30% from the normal scenario. It also shows that with the use of checkpointing method, the makespan increased 13%.

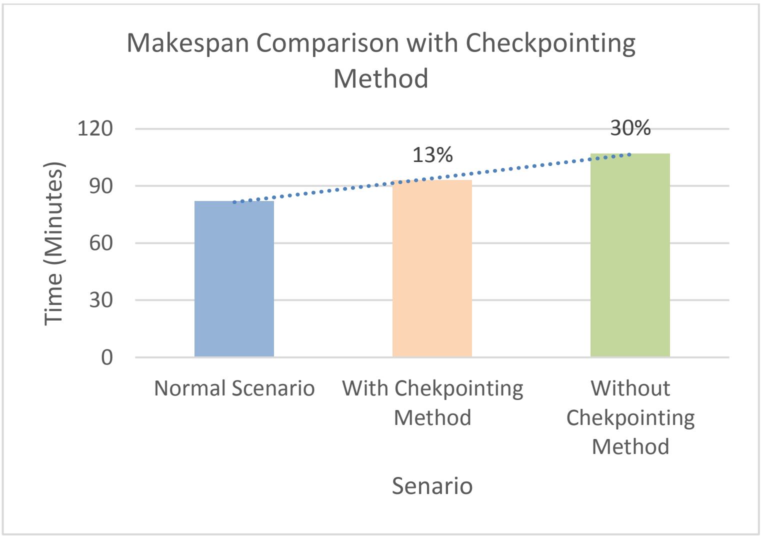

Figure 2. Makespan comparison result for simulation 1

4.2. Simulation 2
In Simulation 1, FAWS algorithm was compared to HLFET algorithm, MCP algorithm and ETF algorithm. In this simulation, FAWS algorithm showed an improvement in the result. The simulation was done using small scale workflow application with three different types of failure handling mechanism. In Simulation 2, FAWS algorithm was compared with HEFT [19] algorithm and CP algorithm increases in parallel to the increase in the size of workflow application in two different conditions. In Simulation 2, the impact of resource failure with recovery mechanism of different workloads was examined. It shows that if the resource restart after failure, the tasks can be distributed to the from 4 resources to 5 resources. Based on the result, when executing workflow application with small number of task (T_{20} and T_{40}), the high number of resources used does not give any effect on the makespan; the makespan of the overall workflow will become stagnant. This will lead to other issues of idle and unused resource that causes increase of cost and waste of energy. In Simulation 2 there are two conditions. Condition 1 the resource experiencing permanent failure and in Condition 2 the resource has temporary failure (restart). The scalability of the proposed algorithm is compared by increasing the number of tasks (T_{60}, T_{80} and T_{100}). Based on the result as shown in Figure 3, the makespan for Condition 1 with permanent resource failure are higher. This is because for large number of tasks, the execution of tasks requires large number of resources. However, by increasing the number of resources it will lead to another problem, that is reliability. The high usage of resources will decrease the reliability of the workflow.

In general, the result of the simulations can be concluded as follows. First, the performance of FAWS depends on the tasks dependencies in the workflow application. The main objective of FAWS algorithm is to identify which task has high number of dependencies. Based on examples, tasks with high dependencies need to be scheduled immediately. Therefore, FAWS algorithm depends with the new ranked list generated from the LWFS algorithm. With the new ranked list, the tasks will be scheduled to designated resources. Second, FAWS algorithm can be used to handle growing number of tasks, from small scale workflow for 10 tasks to large scale workflow of 100 tasks. The result shows the increase of makespan when handling large tasks is about 13% to 15%. The proposed method is scalable to handle large task size and it considerably reduces the failure probability at the expense of relatively minimal increase of the workflow applications makespans.

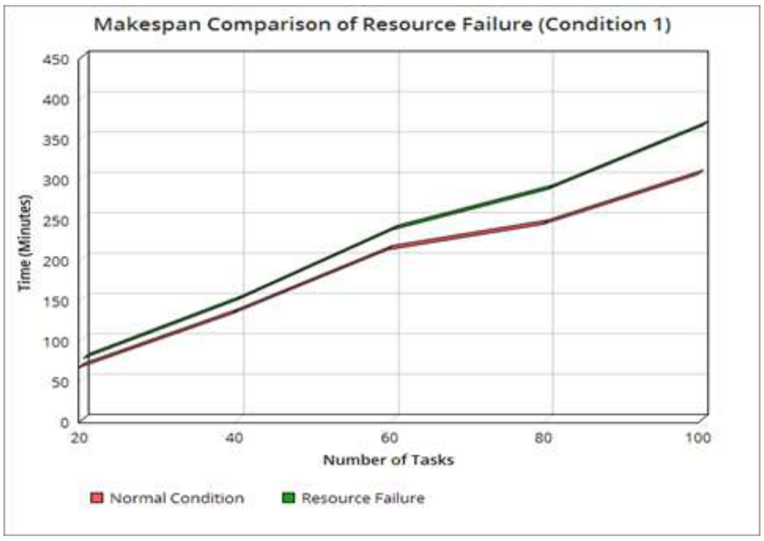

Figure 3. Resource failure condition 1 (permanent)

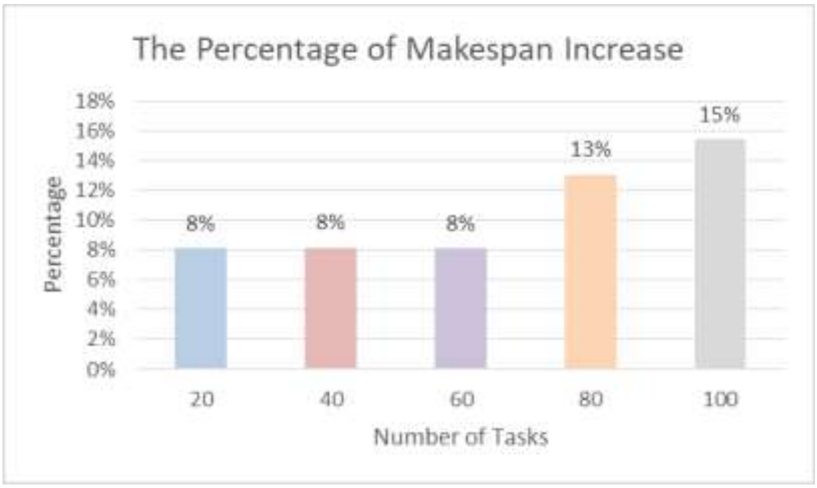

Figure 4. The percentage of makespans during resource failure

For the second of scenario, the result shown in Figure 5 and 6. The result shows trade-off between makespan and reliability of the workflow application. As a result, the makespan and the reliability cannot be minimized simultaneously. The FAWS outperformed both algorithms in terms of minimizing the makespan increase at the same time minimizing the reliability (Figure 5). From this graph, it shows the scalability of FAWS algorithm handles task assignment very efficiently with a minimal increase of the makespan when the number of task increases. It shows that for small number of tasks T_{20}, T_{40} and T_{60}, the increase of makespan is minimal about 8%. As the number of tasks increase T_{80} and T_{100}, the graph shows the relative percent increase in makespan. The result depicts that as the number of task increases, the makespan will also increase. The percentage of makespan gap between the task in conditions of failure and non-failure also increases as number of task increase. It can be seen that when the gap is smaller, it is workflow is more efficient and scalable.

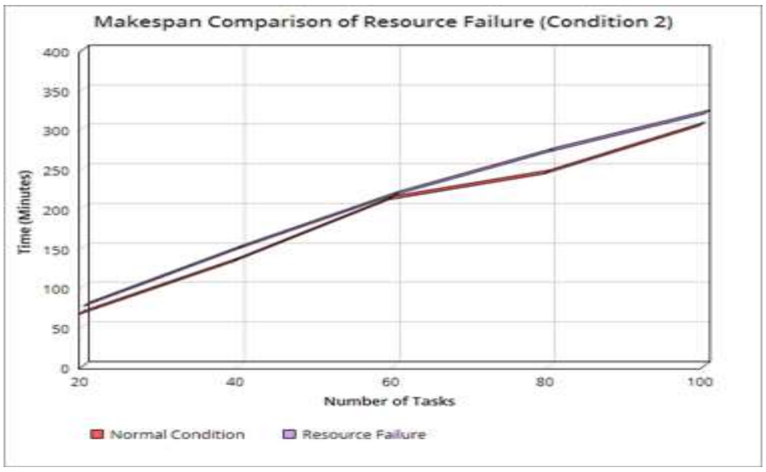

Figure 5. Resource failure condition 2 (restart)

The graph in Figure 6 shows the comparison of makespan of different workflows for both Condition 1 and Condition 2. The result were compared between normal condition of task execution without resource failure presence with the two conditions: permanent and non-permanent resource failure. Each data point represent the number of tasks. The performance of the FAWS algorithms shows variation with respect to the size of the workflow application. When the workflow has fewer tasks, the resources will have more idle time. Hence, the FAWS algorithm will examine and find the next available resources. From this graph (Figure 6), it shows the scalability of FAWS algorithm handles task assignment efficiently with a minimal increase of the makespan.

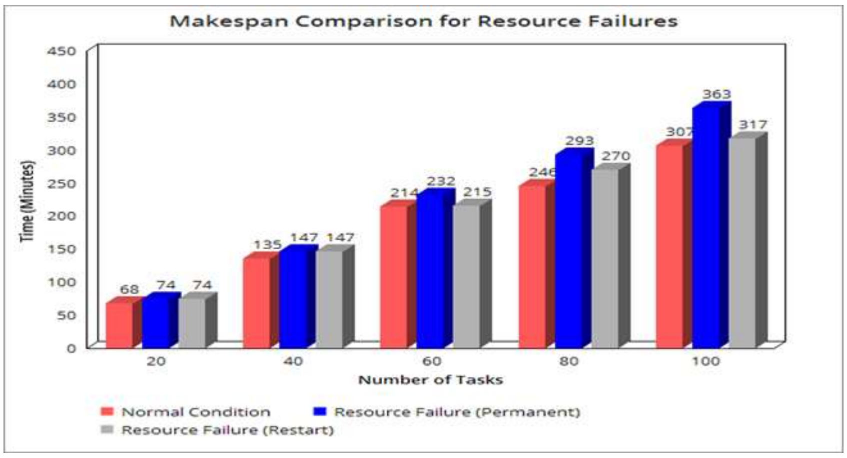

Figure 6. Overall makespans during resource failure

5. CONCLUSION

This paper presented a scalable workflow scheduling algorithm for minimizing makespan and failure probability. The proposed Failure Aware Workflow Scheduling (FAWS) Algorithm handles unexpected failure causes rescheduling of the failed task on the incompleted task execution. To determine the impact on the the makespan in terms of increase of workflow tasks, few experiments were conducted. The algorithm discussed in this paper is for scheduling parallel applications on homogeneous systems solved two conflicting objectives: maximize the reliability and minimize the makespan at the same time. The proposed algorithm handles unsuccessful job execution or resource failure by dynamically scheduling workflows to available resources. The FAWS algorithmis compared with the different scheduling algorithms with an increase of workflow tasks. Based on the experiment, specifically task rescheduling has a huge impact to the subsequent scheduling decisions for not-scheduled-yet task (child) in the workflow. For simulation analysis, task graphs were randomly generated and scheduled the parallel applications on homogeneous systems. The simulation results show that the proposed FAWS algorithm can significantly optimize the makespan and successfully map the workflow tasks to the resources accordingly. The proposed algorithm performs better than existing heuristic-based techniques for scheduling application workflows in terms of efficiency and scalability. One further direction of this research work in this thesis would be to extend the FAWS Algorithm to solve different conflicting QoS objectives such as Cost and Energy. The research will propose a solution that able to minimize the energy consumption at the same time lower the total cost to run a workflow application without sacrificing the system performance.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge Prof Jemal Abawajy from Deakin University, Australia and University Technology Mara, Malaysia. This research was supported by Lestari Grant 600-IRMI/DANA 5/3/LESTARI (0112/2016).

REFERENCES

[1] Pinedo, Michael L. Scheduling: theory, algorithms, and systems. Springer Science & Business Media, 2012.

[2] Balarin, Felice, *et al*. scheduling for embedded real-time systems. *IEEE Design & Test of Computers*, 15.1 (1998): 71-82.

[3] Rodriguez, Maria Alejandra, and Rajkumar Buyya. A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. *Concurrency and Computation: Practice and Experience 29,* no. 8 (2017).

[4] Rahim, S. K. N. A., Bargiela, A., & Qu, R. *Analysis of Backtracking in University Examination Scheduling*. In ECMS, (2013): 782-787.

[5] Zhao, Baoxian, Hakan Aydin, and Dakai Zhu. *Energy management under general task-level reliability constraints.* In Real-Time and Embedded Technology and Applications Symposium (RTAS), (2012): 285-294.

[6] Cardoso, Jorge, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web 1, no. 3 (2004): 281-308.

[7] Khalid, A. N. E., Bakar, A. N., Hu, Yueming, Zhiliang Xie, and Xinda Lu. Approaches to decentralized control of job scheduling for homogeneous and heterogeneous parallel computer systems. *Future Generation Computer Systems* 6.1 (1990): 91-96.

[8] Zuo, Liyun, Lei Shu, Shoubin Dong, Yuanfang Chen, and Li Yan. A multi-objective hybrid cloud resource scheduling method based on deadline and cost constraints. *IEEE Access*5 (2017): 22067-22080.

[9] Zhang, Jinghui, Junzhou Luo, and Fang Dong. Scheduling of scientific workflow in non-dedicated heterogeneous multicluster platform. *Journal of Systems and Software* 86.7 (2013): 1806-1818.

[10] Zur Muehlen, Michael. *Resource modeling in workflow applications*. Proceedings of the 1999 Workflow Management Conference. (1999): Vol. 70.

[11] Zhang, Y., & Chakrabarty, K. *Task feasibility analysis and dynamic voltage scaling in fault-tolerant real-time embedded systems*. In Proceedings of the conference on Design, automation and test in Europe-Volume 2 (2004): 21170

[12] P. Pop, K. Poulsen, V. Izosimov, and P. Eles. *Scheduling and voltage scaling for energy/reliability trade-offs in faulttolerant time-triggered embedded systems*. In Proc. of the Int'l Conference on Hardware/software codesign and System Synthesis, (2007): 233–238.

[13] Workflow Management Coalition (WfMC) Retrieved from URL http://www.wfmc.org

[14] Cardoso, Jorge, et al. Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web 1.3 (2004): 281-308.

[15] Ismail, S. F., & Dout, M. S. N. Multi-objective optimization using Fuzzy evolutionary strategies optimization. *International Journal of Systems Applications, Engineering & Development*, 5(6), (2011): 728-737.

[16] Ismail, A., Yan, J., & Shen, J. Incremental service level agreements violation handling with time impact analysis. *Journal of Systems and Software*, 86(6), (2013): 1530-1544.

[17] Malim, Muhammad Rozi. Lecture timetabling using immune-based algorithms. *JIRKM| Journal of Information Retrieval and Knowledge Managemen*t (2012).

[18] Abdul Aziz, M., Abawajy, J. H., & Chowdhury, M. Scheduling workflow applications with makespan and reliability constraints. *Indonesian journal of electrical engineering and computer science*, *12*(2), (2018): 482-488

[19] Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 2002; 13(3):260–274.