Indonesian Journal of Electrical Engineering and Computer Science
Vol. 12, No. 2, November 2018, pp. 482~488
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v12.i2.pp482-488

Scheduling Workflow Applications with Makespan and Reliability Constraints

Maslina Abdul Aziz, Jemal H. Abawajy, Morshed Chowdhury
Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA Malaysia, Shah Alam, Malaysia
School of Information Technology, Faculty of Science, Engineering and Built Environment,
Deakin University, Geelong, Australia

Article Info

Article history:

Received May 27, 2018
Revised Jul 24, 2018
Accepted Aug 1, 2018

Keywords:

Directed Acyclic Graph (DAG)
Failure-Aware
Makespan
Quality of Service (QoS
Reliability

ABSTRACT

In the last few years, workflows are becoming richer and more complex. Workflow scheduling management system to be robust, flexible with multicriteria scheduling algorithms. It needs to satisfy the Quality of Service (QoS) parameters. However, QoS parameters and workflow system objectives are often contradictory. In our analysis, we derived an efficient strategy to minimize the overall processing time for scheduling workflows modelled by using Directed Acyclic Graph (DAG). We studied the problem of workflow scheduling that lead to optimizing makespan and reliability. The proposed algorithm handles unsuccessful job execution or resource failure by dynamically scheduling workflows to available resources. Based on the experiments results, our proposed Failure-Aware Workflow Scheduling (FAWS) Algorithm can significantly optimize the makespan and minimize the reliability by rescheduling the failed task to the unused resources. The effectiveness of the FAWS algorithm was validated based on a simulation-driven analysis based on the workflow application.

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

Executing a large workflow application is a complex process especially in the situation where there are multiple and different resources involved. It involves complex and highly-structured processes. Every workflow varies individually with different characteristics, priority and dependency. The main motivation of this study is how to find a schedule that manages the execution of different kinds of workflows that have different priorities and interdependent tasks of large datasets without sacrificing the QoS constraints. Identifying tasks that have intermediate tasks and knowing the location of intermediate task are important. Effective scheduling strategies are required in executing large applications to minimize the schedule length (Makespan). Good mapping of tasks to processors in times of failures of resources can have an adverse effect on applications. Therefore, there is an increasing demand for scheduling techniques to minimize the task failure probability at the same time maximize the reliability of during execution of an application. The main challenge is to achieve two objectives; the makespan and reliability that are conflicting.

A workflow is the automation of business processes that involve the processing of cases and the execution of tasks in a particular order, by specific resources, so that some objective is met. It comprised of network, servers, clients and the people need to deal with data and business process rules. For each activity, there will be rules and data that act as controller and input or output respectively. The rules exist due to the

existing of the percentage of influencing by internal and external social and market environment towards an organization [1]. Meanwhile, scheduling can be defined as mapping and managing the execution of tasks on the distributed resources. The challenge would be allocating best-fit resources to workflow tasks while satisfying the objectives set by users. The system performance to store, process and analyze large amounts of distributed data relies heavily on the effectiveness of the scheduling process [2]-[4].

Over the pass years, there are a number of research on different aspect of workflow scheduling such as application, technique, constraint and environment. These papers focus on solving various QoS problems such as system performance, Reliability (R), Energy consumption (E), Time / deadline (T) and cost (C). Some of the papers proposed algorithm using metaheuristic techniques with multi-objective constraints such as genetic algorithm. However, for this research, we only focused on two constraints; time (makespan) and reliability. Our proposed algorithm uses heuristic scheduling algorithms since it resulted positively in solving the scheduling problem with less mathematical formulation. Based on the comparative study, our proposed algorithm differs from existing techniques such as Heterogeneous Earliest Finish Time (HEFT), Modified Critical Path (MCP) and Early Finish Time (EFT). Based on the simulation result, our proposed algorithm resulted positively as compared others.

Automobile and aeronautics are examples of advanced industries which comprises of large complex processes with high uncertainty. These large processes have problems that requires multi-objective solutions to solve [5]. Hwang et al (2003) [6] divided workflow failure handling techniques into two different levels, namely task-level and workflow-level. Unfortunately, as it was proven in many research, when system reliability increases, most of the time, the execution time will also increase. This is because a fast schedule can sometimes be very unreliable.

Many methods have been proposed to deal with faults. One of the alternatives is to have backups and duplications. If let say one of the resources fail, the other resources will continue to operate. However, the major drawback of this method is a possible waste of resources. As an alternative, the scheduling mechanism is introduced. In case of resource failure, the scheduler will decide which resources will start and stop the task. Therefore, the proper use of the scheduling algorithm will minimize the probability of system failure and improve the system response. Scheduling process may lead to reassigning of tasks to resources which could create an infeasible schedule. The more number of data the more complex and difficult to schedule. Therefore, it is important to re-assign the tasks using minimum number of resources [7]-[9]. Unfortunately, as it was proven in many research, when the reliability increases, most of the time, the execution time will also increase. This is because a fast schedule is can sometimes be unreliable. Therefore, there is a need to design an algorithm that look for a set of trade-offs. The objectives are to minimize the makespan and to maximize the reliability of the schedule.

A heterogeneous environment that connects millions of networks and computers is usually at risk to face failures of components/resources (machines, hardware, software and disk) that are located and distributed all over the network. For systems that are critically dependent on computers, for example financial system, these systems have high reliability requirements. Hence, the reliability issues must be considered and dealing with fault-tolerance is a prominent concern especially in large heterogeneous environment where tasks that are related, big, numerous and complex. The process starts when a client request for a service. Once the client's request is submitted, the system needs to be responsive and fast. The client of the service relies heavily on time requirements and constraints imposed by the service provider for the resource capability [10]. A high-performance system is very costly, requires more energy due to the increase of number of resources. However, by adding more resources the system will be less reliable. As an impact, the system will be less secure and cause delay to the overall process, resulting in potentially frustrated clients and lost investors. This will negatively impact business profit and tarnish its reputation.

The main issue at this stage is to have proper positioning of the resources, its relation to other resources and services. When a workflow is submitted submitting to the system, the scheduler determines the best schedule based on the deadline constraints and availability of resources in the system. The challenge we address is how to schedule the W workloads on the R resources but ensuring that each task $t\_i \in T$ is scheduled on one of the $r\_j \in R$ resources. Each $r\_j \in R$ resource executes the tasks without exceeding the deadline D within the maximum use of resources N. Workflow tasks were executed one by one to several distributed resources. Usually, a workflow coordinator handles all workflow tasks execution process. Existing workflow process is unable to handle large number of workflow tasks. Therefore, large number of tasks and large size data will need to be rescheduled to avoid resource failure. This research enhances the existing workflow design [11] by making the following changes:

a) Able to handle large number of tasks when using both single and multiple workflows without sacrificing the QoS constraints.
b) Failure-aware workflow that handles large number of tasks without sacrificing the QoS constraints.

The QoS of service-based systems are becoming increasingly complex (also called CSBS) needs to be maintained during runtime [12]. Any task in a data intensive workflow may handle large-sized data files at both input and output stages of its execution This paper aims to evaluate the performance of the scheduling algorithm for both homogenous and heterogeneous processors. Total execution time is met and therefore increases the performance of the whole system. Thus, the workflow scheduling efficiency and the task execution will improve. Rapidly growing technologies in various areas such as systems, hardware and network. All these technologies aim to give the most efficient and reliable services to a large number of end-users including computing power as a utility, like water, gas and electricity. As discussed earlier, the overall performance of the workflow application relies heavily on makespan constraint especially during runtime. Therefore, by utilizing the resources it will improve the performance of the workflow application.

This research is about enhancing the workflow execution performance by proposing Failure-Aware Workflow Scheduling (FAWS) Algorithm. However, in this paper, we mainly discuss on enabling rescheduling of the failed task to the unused resources to improve the performance. The validate the effectiveness of the proposed algorithm, a simulation-driven analysis based on realistic workflow application was demonstrated. Further, we present a comparative evaluation that looks at two main objectives: (1) utilizing unused resource /application in the event of failure and (2) workflow scheduling against fault-tolerant towards failures. This research tackles the challenges of developing algorithms for rescheduling failed tasks based on the reliability requirement. The simulation results show the FAWS algorithm has optimized the makespan and reliability of the workflow application.

2. FAILURE-AWARE WORKFLOW SCHEDULING (FAWS)

This section continues with a brief discussion of a number of static scheduling algorithm with passive replication strategy. The new proposed scheduling algorithm is named 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) [13]-[14]. This algorithm basic idea is using the execution time of the workflow schedule as the parameter. As shown in the algorithm above, all available processor that is checked on for current backup. The checking process has three main steps.

2.1. Sub Task Scheduling

The tasks T and the backup T' are scheduled on the existing resources. The first step will be estimating the earliest available time for each resource. If a backup task will be slot in any of the resources $R1, R2$ and $R3$ the backup tasks start time will be the earliest task to begin. Then, for the second step is to check the resources start time. If the resources start at the same time, the time gap for the available resources from both overloaded and non-overloaded schedules will be examined. If there are cases where both first and second steps cannot be used to find current backup, the third step will be introduced.

The third step uses makespan as the backup start time. When the earliest execution time to schedule the backup, slot have been found the replication cost of the current backup will be calculated. The replication cost can also be the deciding parameter if in a situation where several resources have the same earliest finish time. The backup task will be scheduled to the resource with minimum replication cost. The performance of FAWS algorithm 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.

Procedure: FAWS

1: INPUT: T task, Prioritizelist, R
2: OUTPUT: Schedule
3: BEGIN
5: **WHILE** ($newList \neq \emptyset$) **DO**
6….Schedule the Primary Copy using HEFT
7….Schedule the Backup Copies
8: $\boldsymbol{t \leftarrow newList}$**.next**
9. $\boldsymbol{r \leftarrow min}$ (r. earlyAvailable)
10. $\boldsymbol{r \leftarrow t}$
11. r. earlyAvailablel += t. s_i
12……T'$_i$ *to* r. earlyAvailablel
13. ENDWHILE
8: Return *Schedule*
8. END FAWS

2.1.1. Case 1: Existing Tasks on Resource with no Failure:

In Case 1, the resources already have tasks scheduled on them. Therefore, we need to find the best resource for the backup tasks. Since there are existing tasks, the backup task need to be schedule overlapping with the existing tasks. But the scheduling process will still follow the four Backup Scheduling Constraints. Our proposed scheduling algorithm is called Best_Resource Algorithm. The main purpose is finding the best resource to place the backup tasks. By scanning the workflow forward from the top or the T_{entry} task to the end, we are able to determine which resources are available to place the backup tasks.

Procedure: Best_Fit_Slot

1: INPUT: T task, Prioritizelist, R
2: OUTPUT: Schedule
3: BEGIN
*5: **For** $(i = 1;\ i \leq |T|;\ i++)$**DO***
4. $list.parent\ [i] \leftarrow\ Parent\ (t_i)$
5. ENDFOR
6: Return list
6. END Best_Fit_Slot

Figure 1 shows the tasks and its backup tasks scheduling. There are 3 resources (R1, R2 and R3) and 8 primary tasks with its respective backups. There is only 1 backup for each task but not all tasks will have a backup. Each backup task will be scheduled after its primary. If the task or the resource fail, immediately the backup task will be activated

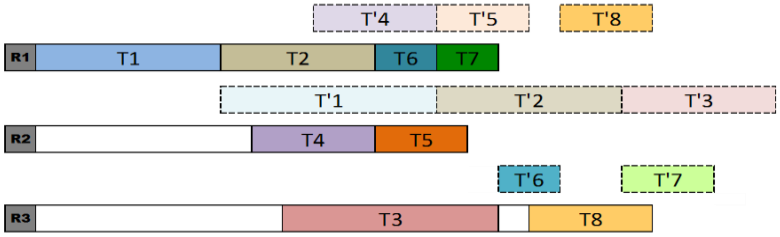

Figure 1. Scheduling backup task with best_resource algorithm

2.1.2. Case 2: Resource Failure

In this case, we propose another algorithm called Best_Fit_Slot Algorithm. Or this example, assume that Resource 1 (R1) fails. In this algorithm, the scanning process goes up starting from T_{exit} task. By looking backwards, we are able to assign backup to an available interval or overlap with existing task without violating the constraints. Assign affected tasks to another resource with the earliest start time. Restart the failed task at the last current position. This algorithm finds the vacant time intervals, either the unoccupied time intervals or overloading with existing primary task.When one of the resources (R1) fails, the backup task will start immediately. As we can see R1 originally, R1 has a full load of 5 tasks. Scheduling the failed task is very challenging since other resources are also packed with tasks. The Best_Fit_Slot Algorithm looks at the load and available time of each resource. This technique also shows the tasks are grouped to save more time. The algorithm also looks at the best way to shift smaller tasks together. The highlighted tasks are the new task queue that combines primary and the backup tasks ready to be executed on other available resources.

Procedure: Best_Resource

1: INPUT: T task,
2: OUTPUT: list
3: BEGIN
*5: **For** $(i = 1;\ i \leq |T|;\ i++)$**DO***
4. $list.child\ [i] \leftarrow\ Children\ (t_i)$
5. ENDFOR
6: Return list
6. END Best_Resource

Figure 2 shows when one of the resources (R1) fails, the backup task will start immediately. As we can see R1 originally, R1 has a full load of 5 tasks. Scheduling the failed task is very challenging since other resources are also packed with tasks. The Best_Fit_Slot Algorithm looks at the load and available time of each resource. This technique also shows the tasks are grouped to save more time. The algorithm also looks at the best way to shift smaller tasks together. The highlighted tasks are the new task queue that combines primary and the backup tasks ready to be executed on other available resources.

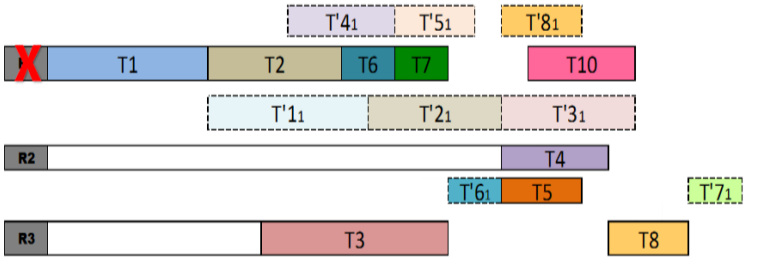

Figure 2. Scheduling backup task with best_fit_slot algorithm

3. RESULTS AND ANALYSIS

The checking process has three main steps. The first step will be estimating the earliest available time for each resource. If the backup can be slot in on this resource, the backup task start time will be set as the earliest start time. The second step is if the resources have the same start time, the resource with the available time slot from both overloaded existing schedules and the time gap among non-overloaded schedules. If there are cases where both steps cannot be used to find current backup, the third step will be introduced. The third step uses makespan as the backup start time. When the earliest execution time to schedule the backup, slot have been found the replication cost of the current backup will be calculated. The replication cost can also be the deciding parameter if in a situation where several resources have the same earliest finish time. The backup task will be scheduled to the processor that has the minimum replication cost. The tasks T and the backup T' are scheduled on the existing resources. When scheduling a new backup, the resource that meets the three conditions as mentioned above. 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 scenarios.

3.1. Scenario 1

If any of the resources fails in the middle of task execution and the task affected is a non-critical task, the tasks will be rescheduled to the next available resources. Figure 3 shows the scenario when resource failure happened at R1.

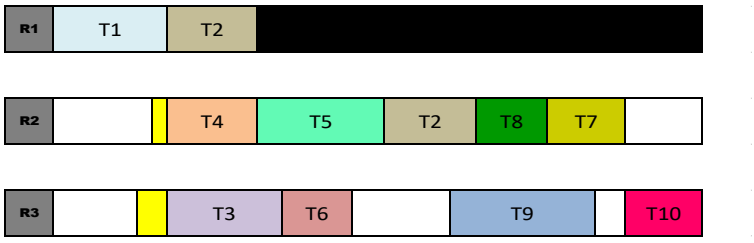

Figure 3. Resource failure in the middle of task execution

3.2. Scenario 2

If the highly dependent Resource R1 fails in the middle of the first task execution (Figure 4). For this type of failure, the rescheduling tasks are not as complicated as in Scenario.

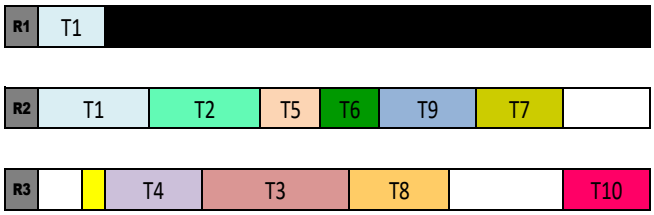

Figure 4. Resource failure in the middle of first task execution

3.2. Scenario 3

For the third scenario, the failure happened at the same spot as in scenario 1. Resource R1 fails in the middle of T_2 execution. Figure 5 shows the failure of R1 when executing task T2 with backups.

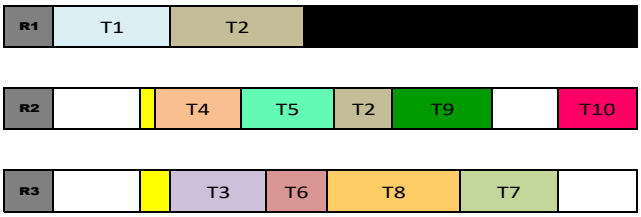

Figure 5. Resource failure in the middle of task execution with backups

By applying our proposed FAWS algorithm, task T2 will not restart from the beginning. Instead, it will continue from the T2 last value when R1 failed. To finish the remaining T2, the resource with the earliest execution time will be chosen. For this situation the remaining time of T2 will start automatically after T5 finishes. The basic idea of this algorithm is to place backup tasks, for tasks that are when a resource fails. This proved that if in a situation where the task fails, the backup task will start immediately from the last value of the failed tasks, without having to restart the whole task. The task scheduling process is the continuation of the Layered Workflow Scheduling Algorithm (LWFS) algorithm [12]-[13]. It uses the same prioritized list generated using LWFS. Based on the experimental result, the overall makespan is 93 mins. This shows, the makespan is less as compared to others. Figure 6 is a graph that summarizes the overall performance of the given scenarios. From this graph, it shows the worst-case scenario is Scenario 1, the ideal scenario is Scenario 3 as compared to the normal situation of workflow scheduling without any failures. For this experiment, we calculated the makespan by simulating it on based on different scenarios using number of resources for the same application. Different scenarios have different effects of the overall makespan. The result can be derived by calculating the difference of the overall time taken for each scenario. Based on the result, it shows that our proposed workflow FAWS improved the makespan of the given workflow. We calculated the makespan by simulating it on based on different scenarios using a number of resources for the same workflow application. Different scenarios have different effects of the overall makespan. The result can be derived by calculating the difference of the overall time taken for each scenario.

Based on the result, it shows that our proposed workflow FAWS increased the makespan from the normal scenario where there is no failure. Since there is task failure during the execution process for one of the resources, the makespan increase is unavoidable. However, the increased makespan is very minimal as compared to others. Figure 7 is a graph that summarizes the overall performance of FAWS based the given scenarios. From this graph, it shows the worst-case scenario is Scenario 1 with 107 minutes, the ideal scenario is Scenario 3 with 93 minutes as compared to the normal situation of workflow scheduling without any failures.

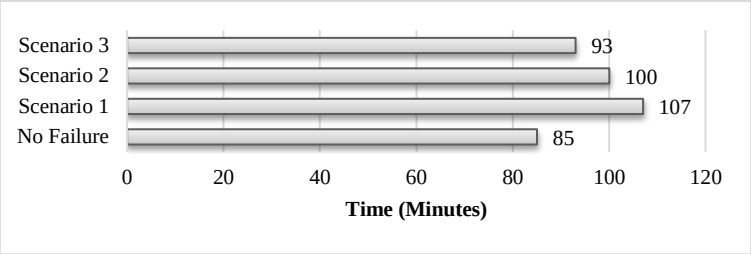

Figure 6. Comparison of makespan for 3 scenarios

4. CONCLUSION

The proposed Failure Aware Workflow Scheduling (FAWS) Algorithm handles unexpected failure causes rescheduling of the failed task on the uncompleted task execution. This is done based on the experiment using DAG graph as a continuation of the Layered Workflow Scheduling Algorithm (LWFS). 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. We compared the FAWS algorithm with the different scheduling algorithms. 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, we randomly generated task graphs 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 is better than existing heuristic-based techniques for scheduling application workflows. For future work, we will look at different conflicting QoS objectives such as Cost and Energy. Since small increase in makespan will have big effect to both constraints, we will propose a solution that able to minimize the energy consumption and cost without sacrificing the system performance.

REFERENCES

[1] Brouns, G. A. J. F. *Featuring workflow management*. Technische Universiteit Eindhoven, Department of Mathematics and Computing Science, 2000.
[2] Wieczorek, Marek, Andreas Hoheisel, and Radu Prodan. *Taxonomies of the multi-criteria grid workflow scheduling problem*. Grid middleware and services. Springer US, 2008. 237-264.
[3] Al-Ali, Rashid, et al. *An OGSA-based quality of service framework*. Grid and Cooperative Computing (2004): 529-540.
[4] Jia Yu. and Buyya R. *A Taxonomy of Workflow Management Systems for Grid Computing.* Journal of Grid Computing, vol. 3, no. 3, pp. 171–200, September 2005. Dongarra, Jack J., Emmanuel Jeannot, Erik Saule, and Zhiao Shi. *Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems*. In Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures, pp. 280-288. ACM, 2007.
[5] Hwang, Soonwook, and Carl Kesselman. *Grid workflow: a flexible failure handling framework for the grid*. In High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on, pp. 126-137. IEEE, 2003.
[6] Malim, M. R. (2012). *Lecture* timetabling using immune-based algorithms. *JIRKM| Journal of Information Retrieval and Knowledge Management*.
[7] Rahim, S. K. N. A., Bargiela, A., & Qu, R. (2013). *Analysis of Backtracking in University Examination Scheduling*. In ECMS (pp. 782-787).
[8] E., Saule, E., & Trystram, D. (2012). *Optimizing performance and reliability on heterogeneous parallel systems: Approximation algorithms and heuristics*. Journal of Parallel and Distributed computing, 72(2), 268-280.
[9] Ismail, A., Yan, J., & Shen, J. (2013). *Incremental service level agreements violation handling with time impact analysis*. Journal of Systems and Software, 86(6), 1530-1544.
[10] Workflow Management Coalition (WfMC) Retrieved from URL http://www.wfmc.org
[11] Ismail, A., & Cardellini, V. (2013, December). *Towards self-adaptation planning for complex service-based systems.* In International Conference on Service-Oriented Computing (pp. 432-444). Springer, Cham.
[12] Aziz, M.A., Abawajy, J. and Herawan, T. *Layered workflow scheduling algorithm*. Fuzzy Systems (FUZZ-IEEE), 2015 IEEE International Conference on. IEEE, 2015.
[13] Aziz, M. A., Abawajy, J., Islam, R., & Herawan, T. *Workflow scheduling on distributed systems*. Industrial Electronics and Applications (ICIEA), 2015 IEEE 10th Conference on IEEE, 2015.