ISSN: 0974-5823 Vol. 7 No. 1 January, 2022

International Journal of Mechanical Engineering

Optimised Universal Workflow Scheduler Using Hybrid Genetic Algorithm

Tejumola Busayo Awolola
Universiti Sultan Zainal Abidin, Kuala Terengganu, Malaysia

Zarina Mohamad
Universiti Sultan Zainal Abidin, Kuala Terengganu, Malaysia

Nor Aida Mahiddin
Universiti Sultan Zainal Abidin, Kuala Terengganu, Malaysia

Wan Nor Shuhadah Wan Nik
Universiti Sultan Zainal Abidin, Kuala Terengganu, Malaysia

Abstract - Technological advancement has accelerated problem-solving in various fields. Cloud computing, a popular technology nowadays, is a type of computing that uses the internet to store and access data and programmes. The advent of cloud computing has helped reduce physical computing infrastructures' cost and maintenance. Many people prefer cloud-based services over expensive and difficult-to-manage local servers, leading to a rapid increase in cloud computing. Tasks are grouped into processes, and a sequence of tasks that process a set of data is referred to as a workflow. As demand for cloud services grows, so do the charges as more workflows are executed. Workflow scheduling was created to improve the efficiency of cloud-based workflow execution. Cloud-based workflow schedulers have a track record of resource management/allocation and task distribution problems. This has led to an increase in execution time and high costs incurred by users. Furthermore, the under-provisioning of resources always affects performance. Many algorithms have been proposed with positive results for various workflow patterns. Depending on the user's goal, different algorithms work efficiently with different workflow problems. Therefore, this study proposed a hybrid genetic algorithm-based resource management system and a cloud computing-optimised workflow scheduler. The genetic algorithm was employed to select the best and most efficient algorithm from a pool of available algorithms that best solves the user's problem. The results obtained indicated that a universal, faster, and more efficient resource management cloud-based workflow scheduler with less execution time and cost-friendliness was achieved.

Index Terms - Cloud computing, genetic algorithm, workflow scheduling, virtual machine

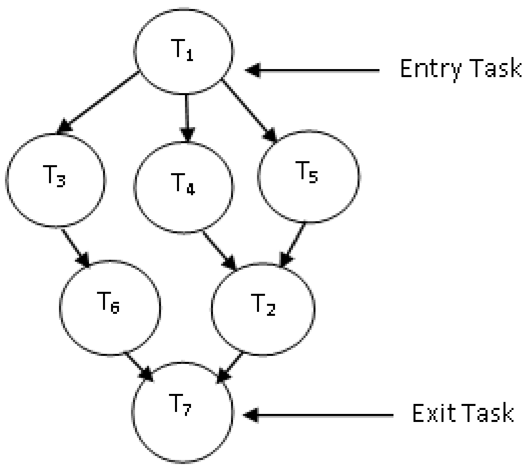
INTRODUCTION

Cloud computing is best known for the accessibility of computer system resources, especially data storage and processing power, from any location and anytime through the internet. Cloud computing is based on the virtualisation model [3], an act of creating virtual computer infrastructures, platforms, and software interfaces to solve various computing tasks. Due to its rapid growth, cloud computing has received a lot of focus lately, and one of its key areas is task and workflow (WF) management.

WF is better expressed as a process or path that describes how a job or task goes from unfinished to completed or from raw to processed. WF has been used to model work practices in different organisations, providing them with a clearer picture of their operations. Furthermore, WF has improved scientific practices by helping in the identification of redundancies, increasing accountability, and reducing micromanagement.

Technological advancement has contributed to a more robust WF with many tasks and processes to be considered. WF scheduler has been the solution to organising and allocating resources to WF processes, thus enhancing WF practice. Thushara [1] showed that WF modelling could be divided into simple and scientific WFs. Figure 1 illustrates how a simple WF using a directed acyclic graph (DAG) task takes place [1].

FIGURE 1: DAG TASK SCHEDULING



Each node represents a task in the figure above, and the edges show the control and data dependency between the tasks. The transfer time is then determined by the data transfer time. On the other hand, the scientific WF model shows different ways scientific applications are modelled, such as epigenomics, laser interferometer gravitational-wave observatory (LIGO) inspiral analysis WF, Montage, CyberShake, and SIPHT, as illustrated in Figure 2.

FIGURE 2: SCIENTIFIC WF APPLICATIONS [1]

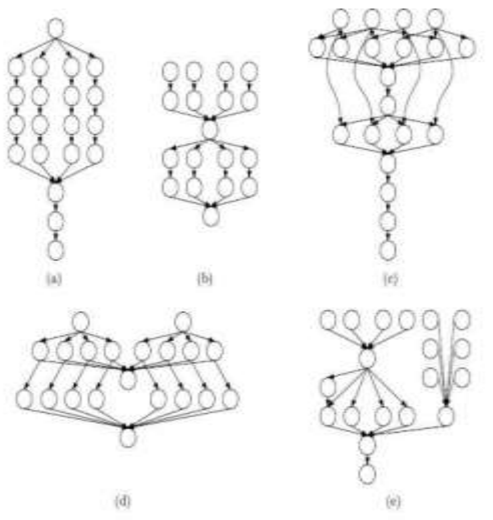


Figure 2 shows different scientific WF diagrams, with each circle representing a task and the arrows representing data dependencies between tasks. Additionally, the structure demonstrates how tasks are executed.

Due to the rapid demand for WF scheduling, the complexity and scalability of its environment have grown significantly, making it a very challenging task. This complexity is inherent in computing WF, thus leading to different scheduling algorithms. Numerous WF scheduling algorithms have been developed with the advent of cloud computing infrastructure as a service (IaaS), owing to the rapid increase in the adoption of cloud services from 5.82 billion (market value) in 2008 to 159.28 billion in 2020 [1].

IaaS provides the necessary computing resources and infrastructure over the internet, such as storage, networking, and virtualisation. With the help of IaaS, users can acquire a virtual machine (VM), which means having a computer in the cloud to run their tasks. This technology allows users to increase or decrease their VM resources anytime. Thus, users are billed based on the resources utilised. The number of tasks to be executed determines the right amount of resources to rent [2]. In other words, renting resources more than required will upsurge the VM cost, whereas having less than necessary will reduce its performance (as the time to run tasks increases). The bi-objective scheduling problem posed by cloud-based WF scheduling is a dominant challenge.

The preceding study highlighted has shown the subject's importance and the need for in-depth thinking while optimising WF scheduling operations. The major problems of a WF scheduler are cost and execution time. Some of the previous studies highlighted earlier have proposed algorithms for resolving them. However, the following issues with their proposed systems are identified. First, their algorithms typically solve one of the two major problems mentioned earlier. Secondly, they focused their efforts on the tasks in the WF rather than on the WF itself. Finally, they used either a single VM with a virtual machine instance (VMI) or distributed VMs with many VMIs, with no flexibility to choose.

In response to the identified problems, this study proposed an optimised hybrid genetic algorithm-based WF scheduler that catered for both cost reduction and fast execution time. Furthermore, the objectives guiding this study are to provide flexibility to WF

scheduler users in terms of the number of VM/VMI to be used for scheduling; to intelligently select the best-fit algorithm for various WF models using a genetic algorithm (GA); to develop an efficient WF scheduling system for both small and large WFs.
Cloudsim, a Java-based framework application used for simulating cloud computing infrastructure and services, was utilised to simulate the proposed algorithm. The application can perform performance tests on the entire system by using its components, such as cloudlet, broker, region, and host. The proposed algorithm and several well-known algorithms were simulated. The findings demonstrated that the proposed algorithm outperformed other algorithms, with a percentage efficiency of 10% for small WFs and up to 40% for large WFs.

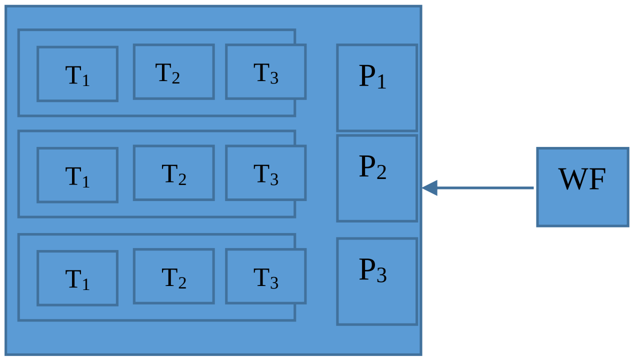
RELATED WORKS

Many studies have been conducted using different technologies to address the prevalent problems highlighted earlier. For instance, the study in [3] suggested reducing the cost and execution time by using a finish time-based algorithm, an upgraded version of the heterogeneous earliest finished time (HEFT) algorithm. The HEFT algorithm was divided into task prioritisation and node selection phases. The first phase (task prioritisation) helps to determine the relative importance of each task depending on its location. Meanwhile, each task will be allocated to a VM during the node selection phase to optimise the cost/time ratio.
Similarly, the authors in [5] proposed an auto-scaling algorithm based on the HEFT algorithm's approach. The study developed an extended version of HEFT, called scalable HEFT (SHEFT), a flexible work scheduling algorithm capable of scheduling WFs in the presence of elastically shifting resources in computing. SHEFT divided the scheduling problem phase into task prioritisation and resource selection. In the first phase, an algorithm is used to prioritise tasks for a WF execution, while the second phase used SHEFT for scheduling WFs in a cloud computing environment. Although SHEFT outperformed HEFT in terms of WF execution time optimisation, the overall target efficiency was not realised.
Another work by [6] proposed a method called gene optimised deep neural round-robin scheduling (GODNRRS) for effectively scheduling cloud-based WFs. This technique consisted of input, hidden, and output layers; each of which interacts with the others to distribute the user tasks optimally among VMs. The study's experiment demonstrated that the method was capable of increasing the efficiency and decreasing the duration of cloud-based WFs scheduling.
More research in this subject led to a study in a multi-cloud domain, in which each provider gives a certain heterogeneous VM number [7]. The study also provided services for storing intermediate data files (called worldwide storage service). Furthermore, the study expressed the scheduler problem as a mixed-integer programme (MIP). The problem highlighted led them to offer algorithm-based solutions for two distinct cases: in the first, the duration unit of each task was considered in hours, while in the second, the duration of each task was assumed to be less than an hour. Initially, the study divided the work in the WF into levels. The assumption was further strengthened by the fact that distinct partitions could not be assigned to separate VMs. This approach made the MIP execution simpler to execute. This method, on the other hand, decreased the optimisation space by restricting the solution space.
Likewise, the study in [8] proposed a clustering approach algorithm for a cloud-based WF scheduler. The tasks levels became the deciding element in the algorithm's scheduling. Further, [9] expanded upon this method by scheduling WF in clouds through a level-based clustering algorithm. Additionally, this study considered each tasks level as a single object (cluster). The algorithm's main purpose was to connect the execution demand, each cluster, and the required resources required for WF execution. Compared to other algorithms, the level-based clustering algorithm improved, on average, the cost, makespan, and the required resources.
GA is another unique algorithm that has been shown to achieve less execution time, cost savings, and efficient scheduling. Many studies have been conducted in its vicinity, some using its robust features and others integrating it with another algorithm. The study in [10] demonstrated how to decrease the duration using a predict earliest first time (PEFT) GA. PEFT was developed for GA with the goal of chromosomes optimisation to produce the most appropriate mutated offspring. Similarly, Ahmed et al. [11] proposed a performance effective genetic algorithm (PEGA). The study demonstrated how optimal results in large space with less time complexity and small makespan could be achieved using a twofold crossover and a single and two-point crossover. These are carried out sequentially to improve the quality and speed of convergence of results.
Besides that, Kaur et al. [12] and Shekar et al. [13] both employed a modified genetic algorithm (MGA) in their respective studies. Kauer et al. [12] proposed the MGA for scheduling tasks in a private cloud to minimise the duration and cost. The initial population was generated in the study using SCFP (smallest cloudlet to fastest processor) and eight random schedules, and the study demonstrated good performance under heavy load. Meanwhile, Shekar et al. [13] used another approach to generate the initial population, specifically using an improved form of max-min to optimise the duration while utilising the same algorithm (MGA). Additional experimentation and comparisons revealed that MGA outperformed GA.
Several other GA-based studies [14-21] that focused on finding an optimal solution to the subject were also reviewed in this study. For instance, the successor concern list heuristics (SCLS) was adopted in [14] to generate the initial population. Furthermore, three hybrid GAs (HGAs) were proposed in [15] to reduce the execution cost of the schedule by using a schedule produced at the bottom and top levels as an initial population. Likewise, the study in [16] proposed an MGA by improving genetic operators. The study proposed a novel fitness function and applied improved genetic operators to obtain an optimum solution. In [17] and [18], a multi-criteria and genetic algorithm were used respectively for job scheduling and load balancing in other to attain quality of service for efficient resource usage. The review in [19] shows the benefit and importance of cloud computing and the need for more research and [20] and [21]
The efficiency of GA in this subject matter cannot be taken lightly since most of the studies done using GA has proved to be efficient in some manner, addressing a problem described by their authors.

HGA

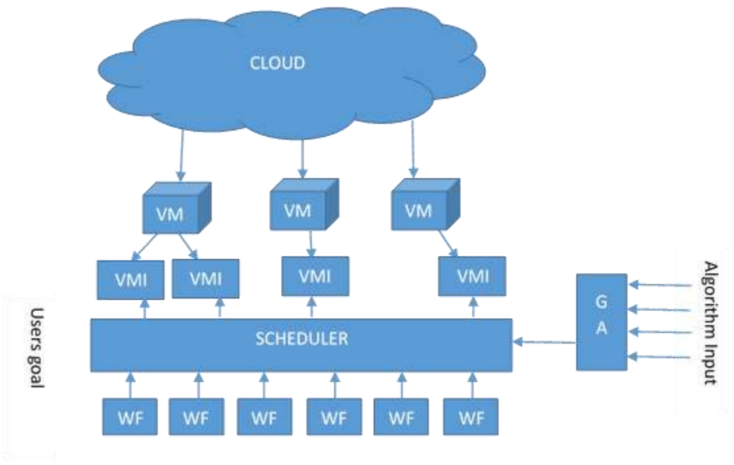
A WF comprises many computing tasks that need to be run using a third-party package. The tasks in cloud WF run on VMI that can be traced to a VM. A VM can have many VMIs to run its task, or a user may use a separate VM for running tasks. Figure 3 below shows the diagrammatic representation of the WF, where T_1, T_2, T_3 are Task 1, 2, 3, respectively and P_1, P_2, P_3 are processes that consist of dependent tasks.

FIGURE 3: A DIAGRAMMATIC REPRESENTATION OF A WF



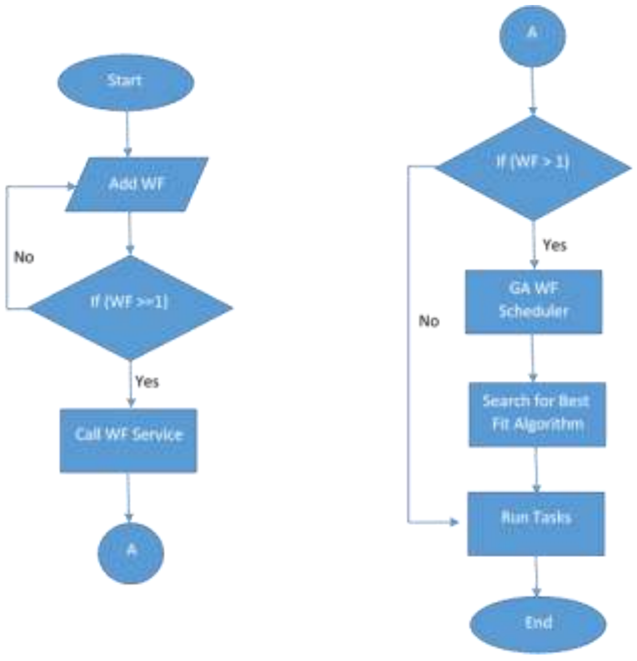
As WF grows, the importance of a scheduler cannot be overstated. Nevertheless, as stated in the literature review, a WF scheduler comes with various challenges. WF scheduler has queued up WF(s) with numerous task(s) awaiting execution. As shown in Figure 4, the proposed system's WF requires VM and VMI to execute the WF's line-up tasks.
This study discovered that different algorithms work efficiently with different WF problems depending on the user's research goal. Therefore, this study proposed a HGA resource management and optimised WF scheduler for cloud computing. HGA was employed to select the best and efficient task receiver from a pool of algorithms that best solve the user's problems. Figure 4 depicts the diagrammatic representation of the proposed system.

FIGURE 4: DIAGRAMMATIC REPRESENTATION OF PROPOSED SYSTEM



As illustrated in Figure 4, the proposed system accepts two user inputs: the WF and the optimisation goal. The two inputs determine which algorithm will be used by the scheduler. The users' optimisation goal is the structure the user wishes to adopt, whether it is a single VM to a WF or a VM with VMI and several WFs based on cost or time optimisation. The process begins when the system receives the required work, as illustrated in the flow chart below.

FIGURE 5: FLOW CHART OF THE PROPOSED SYSTEM



The system allows users to add as many WFs as they wish. When all WFs are added, the call WF service is activated, and the system determines which service to call next by determining if the WF is greater than one. If WF = 1, the task is assigned to VM by default. However, if it is greater than one, the system gives the user the flexibility to assign WF tasks to newly created VM/VMIs, and the WF task is run using the GA-selected best-fit algorithm.

GA Process

GA is a meta-heuristic algorithm influenced by natural selection and genetics in evolutionary ideas. The algorithm follows a basic process, starting with population initialisation and ending with selection and returning the best result. The pseudo-code below shows the proposed system's GA approach.

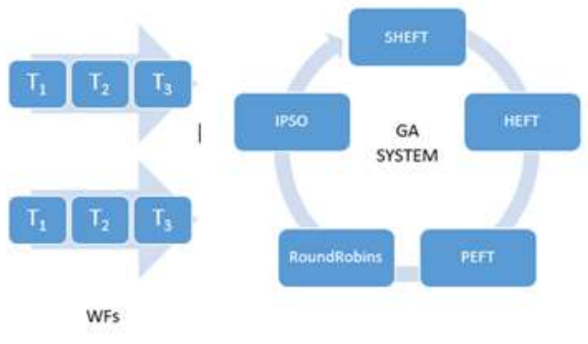
```
Initialise population.

Find the fitness of population
While (terminate criteria is reached)
1.    The parent selection
2.    Crossover with probability pc
3.    Mutation with probability pm
4.    Decode and fitness calculation
5.    Survivor selection
6.    Find best
Return best Algorithms.
```

Population Initialisation

The population is a subset of the current generation's solutions. The proposed system's population is composed of WF, VM/VMI, and available algorithm solutions. In this stage, the scheduler is prepared to process the optimal algorithms solutions for mapping the WF to the VM/VMI for task execution. There are two widely used methods for initialising a GA population: random and heuristics. This study adopted random base initialisation to map the available algorithm to the planned WFs. The next step is to choose the model to adopt in formulating offspring on which the fitness function will operate. Additionally, there are two models: steady-state and generational. The generational model was used in this study, which allows for the generation of "n" offspring depending on the number of available algorithms. The offspring is now ready for the fitness evaluation stage.

FIGURE 6: DIAGRAM OF PROPOSED WF SCHEDULING SYSTEM



Fitness Evaluation

This stage involves the action of a fitness function on the offspring from the previous stage, referred to as chromosomes. This evaluation determines the best algorithm solution for the WF scheduling considering the optimum solution derived after several iterations. While the proposed fitness function evaluates the algorithm's efficiency in terms of the WF's nature, it also takes into account the following factors: time complexity and space/resource management. After evaluating each method, the fitness function assigns a fitness value to each of the algorithms, and other GA processes (parent selection, crossover, mutation) are executed until the optimum solution is obtained.

Selection Process

Several methods can be adopted in selecting the best fit in the selection phase, such as Roulette wheel selection, stochastic universal sampling, tournament selection, and rank selection. This study employed rank selection because of its flexibility in choosing solutions with close fitness value. This selection method now presents the algorithm that can give the best-optimised solution and executes tasks/processes in the WF scheduler.

Simulation

In order to visualise system performance, Cloudsim was used to simulate the process. Some default configurations were considered for the datacentre host and VM, as shown in Table 1. This study considered a single host with a varied number of VMs and WFs. Furthermore, each WF had a constant length, and up to 120 different lengths were generated.

TABLE 1: VM DEFAULT CONFIGURATION

Host Configuration	
VM image size	10000 MB
VM memory	512 MB
VM bandwidth	1000
Operating System	Linux
PES (number of CPUs)	1
MIPs	250
Data centre memory	2048
Data centre bandwidth	1000
Data centre storage per machine	10,000 MB
Data centre number of machine	5
Data centre architecture	X86

CPU: Central Processing Unit

The minimum instruction per second (MIPS) was varied in this study, as shown in Table 2.

TABLE 2: MIPS

VM	**MIPS**
0	200
1	220
2	300
3	450

RESULTS

In order to compute the execution time, this study considered cloudlet characteristics with different lengths and a constant network environment. In addition, each algorithm was tested with a single VM and ten WFs to observe their behaviour.
The same process was repeated for the HGA system. The algorithms' performance was collected considering the number of cloudlets scheduled, execution time, and the scheduling pattern. Then the collected information was used to form a population, crossover rate, and mutation rate.
A fitness function was deployed with respect to the time of execution and cost implication using a base cost of 0.5 RM/s.
The following is the fitness function as represented in [2].

$$Makespan = Max_i\,(TaskFinishtime_i) - Start\ Time \;.... \quad (1)$$

$$Cost = \sum Execution\ times \times resources\ price\ per\ hour + transfer\ costs \;....... \quad (2)$$

The time fitness (F_1)

$$F_1(c) = \frac{1}{Makespan(c)}\ 1 < c < S \;........ \quad (3)$$

And the cost fitness (F_2)

$$F_2(c) = \frac{1}{cost(c)}\ 1 < c < S \;............... \quad (4)$$

Hence, the fitness function is given as below:

$$F_c = \propto f_1(c) + (1 - \propto) f_2(c)\ \alpha \in [0, 1] \;..... \quad (5)$$

$$\propto = \begin{cases} 1 \text{ when only completion time considered} \\ 0 \text{ when only processing cost considered} \end{cases}$$

The following are the proposed GA parameters: a population of 120 individuals (cloudlets) against 3 agents (algorithms); a 50% selection per loop; a 5% mutation ratio.

The following results were obtained for different algorithms: HGA, FCFS (first come first serve), SJF (shortest job first), and RR (round robin).

TABLE 3: AVERAGE PROCESSING TIME FOR TEN WFs

No. of VM	No. of WF	RR	SJF	FCFS	HGA
1	10	68.83	68.83	68.83	68.83
2	10	57.08	55.38	56.56	57.08
3	10	20.38	20.38	22.40	20.38
4	10	13.86	15.86	16.88	13.86

TABLE 4 AVERAGE PROCESSING TIME FOR 20 WFS

No. of VM	No. of WF	RR	SJF	FCFS	HGA
1	20	147.42	147.42	147.42	147.42
2	20	57.08	55.92	55.92	55.92
3	20	37.71	37.71	39.91	37.71
4	20	26.11	28.11	29.21	26.11

TABLE 5: AVERAGE PROCESSING TIME FOR 40 WFS

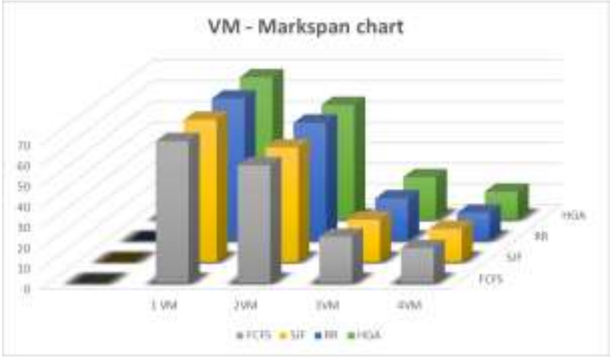
No. of VM	No. of WF	RR	SJF	FCFS	HGA
1	40	280.08	280.08	280.08	280.08
2	40	110.52	108.30	108.32	108.30
3	40	71.99	71.99	73.99	71.99
4	40	49.45	51.48	52.68	49.45

TABLE 6: AVERAGE PROCESSING TIME FOR 80 WFS

VM No.	No. of WF	RR	SJF	FCFS	HGA
1	80	577.54	577.54	577.54	577.54
2	80	226.83	210.33	214.63	210.33
3	80	146.26	146.26	158.46	146.26
4	80	101.22	113.22	115.43	101.22

The tabulated results above were obtained through Cloudsim simulation. The data show the makespan values of each algorithm against the proposed algorithm (HGA), and the pattern is represented with a chart as shown in Figure 7.

FIGURE 7: VM MAKESPAN PATTERN USING TABLE 3



The above illustration was derived using data from Table 3. Similarly, all other tables exhibited a similar pattern. The results revealed that the makespan for a single VM was identical across all algorithms, proving that a VM scheduling algorithm is unnecessary when only one VM is available in a data centre.

Additionally, increasing VM number resulted in a decrease in computing time as some WFs were moved to the next available VM. The increased VM number is intended to assist in the processing of large WFs, which necessitates the need for a scheduling algorithm to optimise the WF to VM allocation.

Moreover, as shown in Table 4, there is a slight change in computing speeds when using different algorithms as the number of VMs increases. This difference indicated that some algorithms perform better with fewer VMs, while others perform better with a larger number of VMs.

Numerous experiments were conducted to evaluate the performance of the proposed algorithm and all of its components: testing the aggregated multi-objective fitness function when multiple quality of service (QoS) metrics are required; testing the penalty method and the viral infection operator when QoS and hard constraints are added; comparing the proposed algorithm's convergence speed to others; conducting some experiments when scaling resources. Except for scaling resources, all results used a pool of one resource per type (total three resources).

The following are the QoS variables considered in the experiments:

1. **Makespan:** The makespan is the total time needed to execute a WF application. It is given in Equation (1) and calculated by adding the different execution times of the tasks on the resources and the data transfer times between the resources.
2. **Cost:** The overall cost, including resource use and data transfer (between them) costs, per hour. It is given in Equation (2).
3. **Reliability:** The probability that the WF schedule will be completed successfully without any resource failure. The model for reliability in this study was inspired by [12] and based on a resource-specific failure rate λi. It was derived from Equation (6).
4. **Availability:** The global availability of the rented resource pool, which can be used to determine the extent to which all rented resources are utilised. It was determined using Equation (7), where N denotes the pool's total number of resources.

$$Reliability = \exp - \sum \lambda i * execution\ time \ldots (6)$$

$$Availability = \frac{1}{N} * \sum N - executiontime_i\ Makespan\ \ldots\ldots.(7)$$

The proposed system was able to address the issue of cost reduction with fast execution time, enable users to optimise their system performance by increasing the number VM, use the best-fit algorithm for scheduling, and can be adopted by small and large workflows.

CONCLUSION AND FUTURE WORK

The goal of this study was to achieve a universally optimised WF in terms of cost, resource allocation, and time efficiency while taking into account the different WF patterns adopted by users. The proposed system enables complex WFs to be scheduled efficiently and effectively using the best available algorithm, although additional time is required to compute a solution using the best algorithm.

Appendix

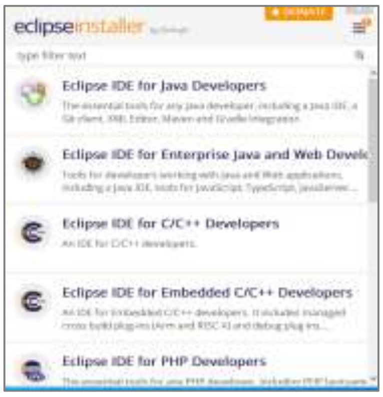
Installing CloudSim

A Java IDE (integrated development environment) is required to run CloudSim. Eclipse and NetBeans are the best and most popular Java IDEs. Eclipse IDE was used for this research. The steps below outline how to install the Java IDE.

Step 1: Download Eclipse, CloudSim, and common-Math3-3.6.3 from the below links and put them in a single folder:

- https://eclipse.org/downloads
- https://github.com/Cloudslab/cloudsim/releases/tag/cloudsim-3.0.3
- http://www.java2s.com/example/jar/c/download-commonsmath3361jar-file.html

Step 2: Run Eclipse setup and select Eclipse for Java Developers, then allow it to download the basic Java libraries to get started as shown below.



eclipseinstaller

type filter text

Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration

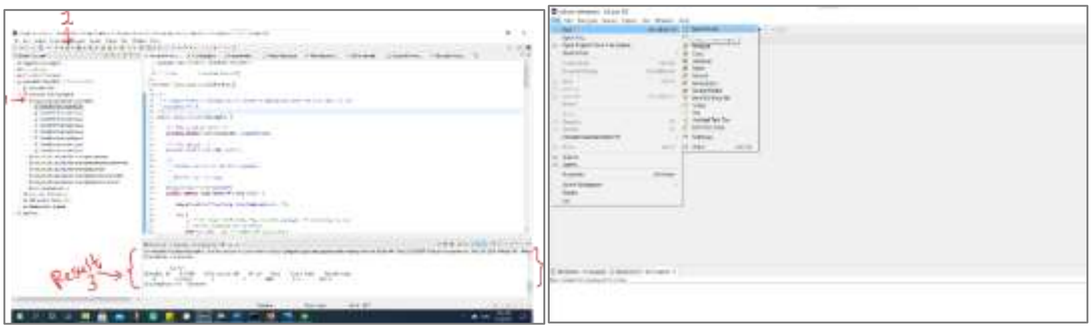
Eclipse IDE for Enterprise Java and Web Develo
Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer...

Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers.

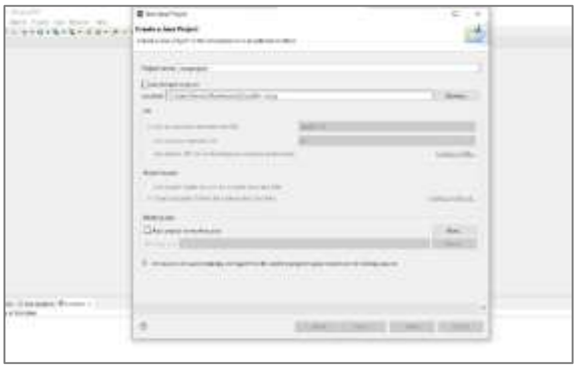
Eclipse IDE for Embedded C/C++ Developers
An IDE for Embedded C/C++ developers. It includes managed cross build plug-ins (Arm and RISC-V) and debug plug-ins...

Eclipse IDE for PHP Developers

Step 3: Create a new project as shown below:

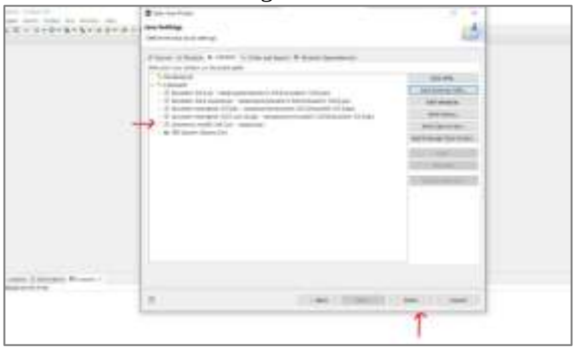


Step 4: Uncheck the default location and navigate to your extracted CloudSim folder as your new location as shown below:



Click on 'Next' to continue

Step 5: On the next page, select the libraries tab and load the common maths jar file from ‘Add External Jars’. Navigate to where you have the file, select it, and click on done as shown in the figure below:



After the above steps, you can proceed by clicking on the finish button.

Running your first project

On the right side of the IDE is the Project Explorer. It displays the folders and files used in the project. Navigate to cloudsim 3.0.3/examples.

Select the example of your choice by double clicking on it to load it into the workspace. Click on run (labelled 2) below and wait for the code to be fully compiled. The result is shown in the console (labelled 3), as shown below.

Acknowledgment

This work is supported by Fundamental Research Grant Scheme (FRGS/1/2018/ICT03/UNISZA/02/01) under the The Ministry of Higher Education (MOHE) and Universiti Sultan Zainal Abidin (UniSZA), Malaysia.

Conflicts of Interest

There are no conflicts of interest to disclose on the part of the authors

REFERENCES

[1]. A. Thushara. Scientific Workflow Scheduling in Cloud Computing Environment: A Survey. International Journal of Computer Engineering & Technology (IJCET). 2018; 9(6): 83 - 9.
[2]. A. H. a. Z. M. Tawfiq Alrawashdeh. Scientific Workflow Scheduling in Clouds: A Review. International Journal of Engineering & Technology. 2018; 7(3.28): 271-274.
[3]. N. D. &. H. E. N. Man. Cost and efficiency-based scheduling on a general framework combining between cloud computing and local thick client. Proceedings of the IEEE international conference on computing. 2013; 258-263.
[4]. Z. C. G. &. F. W. Jiping. HEFT based cloud auto-scaling algorithm with budget constraint. International Journal of Advance computer Science and Technology. 2014; 3: 13 - 18.
[5]. C. L. a. S. Lu. SHEFT: An Elastic Workflow Scheduling Algorithm for cloud computing. 2011.
[6]. K. H. M. a. K. R. Shanmugasundaram M. Gene Optimized Deep Neural Round Robin. International Journal of Advanced Computer Science and Applications. 2019; 10: 337 – 346.
[7]. M. F. K. B. M. D. E. &. N. J. Malawski. Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization. Scientific Programming, 2015; pp. 1 - 13,
[8]. D. A. L. B. S. G. V. R. S. M. P. N. C. M. G. S. A. L. N. &. K. S. Prathibha, Efficient scheduling of workflow in cloud environment using billing model aware task clustering. Journal of Theoretical and Applied Information. 2014; 65(3): 595-605,.
[9]. Z. M. a. A. H. Z. Tawfiq Alrawashdeh, Level Based Clustering Approach to Scheduling Workflows in clouds. International Journal of Engineering and Technology. 2018; 7(3.28): 284 - 289.
[10]. D. K. G. a. R. M. S. Rohit Nagar. Time Effective Workflow Scheduling using Genetic Algorithm in Cloud Computing. I.J. Information Technology and Computer Science. 2018; 1: 68 - 75
[11]. M. U. N. W. A. Q. a. C. W. Ahmad G.S, "A Performance Effective Genetic Algorithm for Task Scheduling in Heterogeneous System," IEEE International Conference on High Performance Computing and Communication.2012; 1082 -1087
[12]. V. Kaur S. An Efficient Approach to Genetic Algorithm for Task Scheduling in Cloud Computing Environment. International Journal of Information Technology and Computer Science. 2012; 74 - 79,
[13]. S. S. a. M. Kalra. Scheduling of Independent tasks in cloud computing using Modified Genetic Algorithm. IEEE, no. 10.1109/CICN. 2014;128: 565 -569.
[14]. J. G. Y. W. a. T. Z. Chuan Wang. Hybrid Heuristic-Genetic Algorithm for Task Scheduling in Heterogeneous Multi-Core System (HSCGS). Springer, no. DOI: 10.1007/978-3-642-33078-0_12. 2012.
[15]. A. V. a. S. Kaushal. Deadline constraint heuristic-based Genetic Algorithm for Workflow Scheduling in Cloud. IEEE, 2014; 5 (2): 96 -106.
[16]. B. Z. a. Hongze. Modified Genetic Algorithm for DAG Scheduling in Grid System. IEEE. 2012; 465 - 468.
[17]. M. Zarina, N.A Mahidin, Nor Aida Mahiddin. A genetic algorithm for optimal job scheduling and load balancing in cloud computing. International journal of Engineering and Technology (UAE).2018; 7(3.28(28)):290-294.

[18]. Aminu A. M., Zarina M., Wan Nor Shuhadah Wan Nik, Fadhilah A., Mohamad A. M. and Mustafa M. D. A Random Make Genetic Optimizer for Resource Maintenance In Cloud Computing. World Applied Sciences Journal 35 (New Advancement of Research & Development in Computer Science): 2017, 137-146..
[19]. Tawfiq A, Azinda H. Z., Zarina M, . Scientific Workflow Scheduling in Clouds: A Review. International Journal of Engineering & Technology, 7 (3.28) (2018) 271-274..
[20]. Ibrahim, I. MTask scheduling algorithms in cloud computing: A review. Turkish Journal of Computer and Mathematics Education (TURCOMAT). 2021; 12(4): 1041-1053.
[21]. Bandaranayake, K. M. S. U., Jayasena, K. P. N., & Kumara, B. T. G. S. An Efficient Task Scheduling Algorithm using Total Resource Execution Time Aware Algorithm in Cloud Computing. In 2020 IEEE International Conference on Smart Cloud (SmartCloud) 2020; 29-34.
[22]. Gaetan H., Daniel de Oliveira, Esther P. Christophe P. Francois T., and Patrick V. Cache-aware scheduling of scientific workflows in a multisite cloud. Future Generation Computer Systems 2021; 172 - 168.