International Journal of Advanced Technology and Engineering Exploration, Vol 11(111)
ISSN (Print): 2394-5443 ISSN (Online): 2394-7454
http://dx.doi.org/10.19101/IJATEE.2023.10102108

Research Article

Workflow scheduler optimization using an enhanced hybrid genetic algorithm

Awolola Tejumola Busayo[1], Zarina Mohamad[1*], Nor Aida Mahiddin[2] and Wan Nor Shuhadah Wan Nik[1]
Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, [UNISZA] Kuala Terengganu, Malaysia[1]
East Coast Environmental Research Institute (ESERI), Universiti Sultan Zainal Abidin, [UNISZA] Kuala Terengganu, Malaysia[2]

Received: 12-August-2023; Revised: 11-February-2024; Accepted: 13-February-2024;

Abstract
The effectiveness of genetic algorithms (GA) can be improved by adjusting genetic operators and integrating an efficient heuristic. These enhancements are integrated into the suggested enhanced hybrid genetic algorithm (e-HGA). The e-HGA begins with an initial population that includes a solution derived from a heuristic, which serves as a guiding point toward achieving an optimal makespan solution. The proposed e-HGA was evaluated in this work for two degrees of fitness, which qualified a chromosome and a gene to be preferred above their other counterpart in a data population. To preserve population variety and avoid premature convergence, parents were randomly picked from the population and crossed over (mated) to generate offspring that were then modified by introducing random geneLists. The conventional hybrid genetic algorithm (HGA) and e-HGA required 9.95 s and 9.148 s, respectively, for task completion. Increasing the number of cloudlets to 40, the conventional HGA and e-HGA took 10.674 s and 9.558 s, respectively. When 50 cloudlets were assigned to 10 virtual machines (VMs) the conventional HGA completed the task in 11.01 s, while the e-HGA required 12.863 s. Subsequently, with 60 cloudlets on 10 VMs, the conventional HGA and e-HGA achieved task completion in 14.74 s and 14.242 s, respectively. For 70 cloudlets on 10 VMs, the conventional HGA and e-HGA required 15.38 s and 17.25 s, respectively. The results contributed to research on task scheduling optimization by scheduling task operations to reduce cost, enable efficient resource allocation, and manage time.

Keywords
Genetic algorithm, Enhanced hybrid genetic algorithm, Hybrid genetic algorithm, Scheduling, Makespan, GeneLists.

1.Introduction
To increase efficiency, an enhanced hybrid genetic algorithm (e-HGA) for task scheduling is created by integrating genetic algorithm (GA) with additional optimization approaches. Task scheduling is a difficult problem, especially in parallel and distributed systems. A hybrid method allows the strengths of many algorithms to complement one another, resulting in superior results [1]. Task scheduling in distributed computing environments is a critical aspect that directly impacts the overall performance and efficiency of the system. Efficient task scheduling ensures that computational resources are utilized optimally, leading to improved system throughput; reduced completion times, and enhanced overall system reliability. In this context, an e-HGA can be a powerful tool for addressing the complexities of task scheduling. This approach combines the strengths of GA with other optimization techniques, making it well-suited for the complex and dynamic nature of distributed computing environments [2].

In recent times, there has been an unparalleled advancement in cloud computing technology, coinciding with the proliferation of complex and interconnected data that form scientific workflows (SWFs). This has significantly underscored the importance of SWFs which has become a top priority for both service providers, and customers [3]. Consequently, efforts have been directed towards devising optimal strategies for assigning workflow tasks to available computing resources [4]. Cloud computing has emerged as a prominent research domain, gaining recognition as the primary distributed computing model. It offers on-demand, scalable, and highly reliable resources, operating on a subscription-based service model kin to utility

*Author for correspondence

computing. These resources are utilized for executing SWFs, which include notable examples like Montage, CyberShake, Epigenomics, laser interferometer gravitational-wave observatory (LIGO), and Stanford Information prediction heterogeneous tools (SIPHT). SWFs can be viewed as specialized workflows designed for computational processes involving intricate data flows and control dependencies [5]. They automate the implementation of these processes on suitable computing resources. The current era is characterized by remarkable advancements in cloud computing technology, complemented by the increasing complexity of SWFs; this has necessitated a strong emphasis on efficient workflow scheduling (WS) benefiting both providers and customers alike. Cloud computing has emerged as a leading model for distributed computing, offering flexible and reliable resources for executing SWFs which play a pivotal role in various domains such as Montage, CyberShake, Epigenomics, LIGO, and SIPHT. To ensure the successful execution of SWFs, it is crucial to utilize available computing resources optimally. This involves focusing on identifying the optimal method for allocating workflow tasks among these resources, a procedure referred to as WS. WS encompasses the assignment and oversight of the execution of interconnected tasks, all while considering constraints on shared resources based on their priorities [6]. The combinatorial nature of this problem makes it known to be nondeterministic polynomial (NP) complete [6], prompting researchers to seek near-optimal solutions. In order to facilitate the execution of workflows, it is essential to have well-defined and effectively managed workflows. This is where an efficient workflow management system (WMS) comes into play. A WMS is responsible for defining and organizing workflows in a manner that allows for their subsequent execution. By utilizing robust WMS, researchers can ensure that workflows are efficiently managed and prepared for execution. The basic cloud computing concept is to isolate applications, hardware, and the operating system. For example, if an operating system failure or a virus attack occurs, virtualization technology can migrate the application automatically to another server instead of shutting down the whole system. One physical server can host several virtual servers. Furthermore, one cloud user can possess ≥ 1 virtual instance for data storage or hosting on cloud servers. Job scheduling maps jobs to the available task resources in a cloud computing environment. Scheduling involves mapping and managing the execution of interdependent tasks on allocated resources [7]. WS represents a widely studied problem in computer science, characterized as a NP hard challenge. Researchers have devoted substantial effort to enhancing the performance of workflow execution [8]. NP-hard problems are those that can be transformed into different problems solvable in polynomial time on a nondeterministic machine [8]. This category encompasses optimization problems like the fractional knapsack and the traveling salesman. The effectiveness of virtual machines' (VMs) performance relies on factors such as processor capabilities, memory capacity, and processing architecture. While a VMs equipped with high-performing processors and ample memory has the potential to deliver superior results, these resources must be judiciously managed to avoid inefficiencies stemming from suboptimal workflow structuring [9]. Consequently, the optimization of WS becomes pivotal for allocating tasks within both single and multiple VMs in cloud environments. The primary objective of researchers has been to optimize cloud WS systems in terms of both time and cost. Numerous methodologies have been adopted to tackle this multifaceted problem. This study centers on refining the scheduling of workflows onto VM aiming to achieve resource efficiency and subsequently reduce infrastructure costs [10]. This research is mainly aimed to allocate appropriate resources to workflow tasks to enable the completion of workflow task execution within the customer's stated deadline. An appropriate scheduling strategy can substantially influence cloud computing performance [11]. Scheduling aids resource organization as specified by the user. Resources must be formally allocated prior to scheduling for application execution. The processor, memory, and workflow processing architecture determine VMs performance. Processing time and cost reduction are major issues in tasks running on VMs. Makespan (the maximum time one VM requires to complete all assigned tasks or jobs) is an important cloud computing issue. The GA represents a probabilistic approach for conducting a comprehensive search across possibilities, inspired by the principles of natural biological evolution. GAs function by manipulating a population of potential solutions using the survival of the fittest principle aimed at progressively improving the quality of approximations to a solution. In each of the iteration a fresh collection of approximations is generated by picking individuals according to their degree of suitability within the problem domain and then combining them using operators inspired by biological genetics. In this study, the research issue

focused on how scheduled tasks can yield minimal job completion times using a proposed e-HGA. Grid computing and cloud computing resources offer optimal solutions that can cater to user requirements, providing scalability and flexibility for the considered applications [12]. However, there are distinct differences in task scheduling between cloud computing and grid computing: (1) Resource sharing: Cloud computing leverages advanced services by utilizing resource sharing through virtualization technologies and internet-based concepts [13]. This enables real-time allocation, maximizing resource utilization and enhancing the elasticity of cloud services. Consequently, the scheduler in a cloud workflow system must consider the virtualization infrastructure, such as virtual services and VM to effectively support computational processes. On a contrasting note, grid computing relies on the collective distribution of an extensive collection of resources. Its main emphasis is on batch processing, where resources become available as they are freed up by other users. (2) Regarding the expense associated with resource utilization: Cloud computing offers a versatile cost structure that takes into account the user's specific requirements, encompassing options like pay-as-you-go and on-demand services. This approach empowers users to be charged in accordance with the resources they utilize, tailored to their individual needs. In contrast, grid computing follows a quota-based strategy to determine the accumulated cost of requested services [14]. Grid computing lacks the flexible costing mechanism present in cloud computing. In summary, cloud computing and grid computing differ in terms of resource sharing and cost of resource usage. Cloud computing emphasizes resource sharing through virtualization to support real-time allocation and scalability, while grid computing focuses on shared resource clusters and follows a quota-based costing strategy. To efficiently schedule and map workflow tasks to the available resources within a cloud environment, a workflow scheduler (referred to as a "bridge" in *Figure 1*) is required.

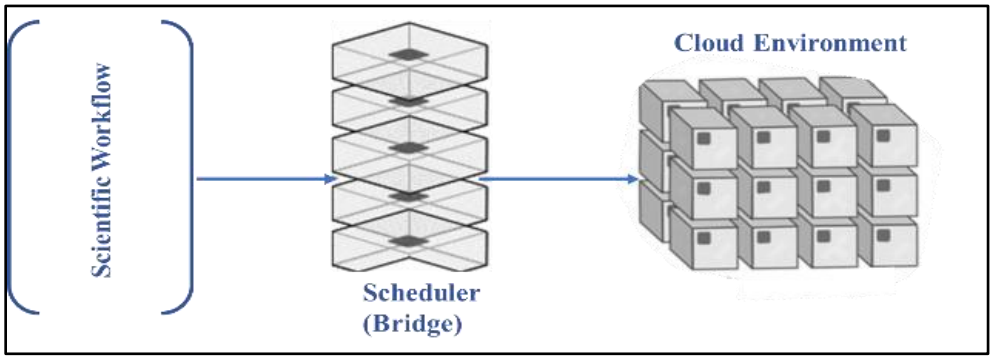

Figure 1 The architecture for executing scientific workflows in a cloud environment

Multi-objective optimization: WS often involves multiple conflicting objectives, such as minimizing execution time while minimizing costs. GA can handle multi-objective optimization problems effectively by finding trade-off solutions. Green computing: Energy efficiency is a growing concern in data centers and cloud computing. An e-HGA can help optimize scheduling to reduce energy consumption, contributing to environmentally sustainable computing practices. Real-world applications: WS optimization has applications in various domains, including scientific research, healthcare, finance, and logistics. Improving scheduling algorithms can lead to advancements in these fields. In summary, the motivation for researching workflow scheduler optimization using an e-HGA is driven by the need to address the challenges posed by complex, large- scale workflow environments and the potential benefits in terms of resource utilization, cost savings, scalability, and improved quality of service (QoS) across various industries and applications. The primary objective of researchers has been to optimize cloud WS systems in terms of both time and cost. Numerous methodologies have been adopted to tackle this multifaceted problem. This study centers on refining the scheduling of workflows onto VMs, aiming to achieve resource efficiency and subsequently reduce infrastructure costs.

WS is a NP hard problem that researchers are studying to improve workflow execution performance. NP-hard problems can be reduced to different problems using polynomial time on nondeterministic machines, such as optimization problems or fractional knapsacks. VM performance is

determined by processor, memory, and processing architecture. Optimizing WS for task allocation in single and multiple VM on the cloud is a major task for researchers. This work focuses on scheduling workflows to efficiently manage resources and provide a less cost infrastructure. This research area aims to address the challenges associated with efficiently managing and scheduling tasks in workflow environments, such as cloud computing, data centers, scientific simulations, and manufacturing processes. Below are some key motivations for this research: Resource utilization: Efficiently allocating and utilizing computing resources is crucial to maximize the throughput and minimize operational costs. An e-HGA can help find optimal resource allocations for workflow tasks, ensuring that resources are used effectively. Time and cost savings: Workflow optimization can lead to significant time and cost savings. By automating the scheduling process and improving its accuracy, organizations can reduce execution times and operational expenses. QoS Many applications, particularly in cloud computing and real-time systems, require adherence to specific QoS constraints. Optimizing WS ensures that these constraints are met consistently.

The paper is organized as follows: Section 2 provides a comprehensive review of the literature related to the study of WS, WS objectives, and task scheduling algorithms for optimizing workflow schedulers using an e-HGA. Section 3 describes the methodology of the study, details the proposed hybrid model, and presents the experimental study conducted. The results obtained from these experiments are analyzed and compared with other models in Section 4. Section 5 discusses the overall analysis of the results and their impact. Finally, Section 6 concludes the paper and discusses future research directions.

2.Literature review

2.1Workflow scheduling

Effective scheduling is vital for optimum workflow execution on all execution platforms. Scheduling enhances performance by capitalizing on workflow parallelism content. Nevertheless, Amdahl's law states that sequential workflow limits the advantage of parallelism. Processing scheduling stands as a pivotal subject within cloud computing, focusing on the efficient execution of processes while giving due consideration to QoS prerequisites like time limits and financial constraints. In scholarly works, numerous state-of-the-art algorithms for scheduling workflows are fundamental or research-oriented, in the realm of cloud computing, have been formulated. Cloud computing, a technological advancement, furnishes expandable services to users by harnessing remote centralized computers and the internet [15]. It leverages a diverse range of distributed resources to deliver various services, each with its own unique QoS requirements [12]. Amazon elastic compute cloud (EC2), GoGrid, Google App Engine, Microsoft Azure, and Aneka are some of the most well-known cloud computing systems. Public clouds, private clouds, community clouds, hybrid clouds, and cloud federations are the most common types of clouds [16]. Public clouds are open to any user [17], while private clouds are exclusively owned and accessed by specific enterprises [16]. Community clouds are shared among multiple organizations and can be managed either by those entities or by external service providers [16]. Hybrid clouds merge resources from both public and private cloud sources [18]. Moreover, the concept of multi-cloud configurations [19, 20]; Has emerged to address availability challenges by integrating separate cloud environments. Cloud services are delivered through software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) providers [21]. SaaS providers lease corporate SaaS of clients [22], PaaS suppliers provide web-based access to development components [22], and IaaS clouds offer fundamental cloud infrastructure resources such as computation power, storage, and networking [23]. Virtualization plays a pivotal role in facilitating cloud computing by enabling multiple VMs to coexist on a single physical computer [24]. Each VM simulates an independent computer system and executes tasks assigned by users [25]. Through VMs instantiation, users can deploy their applications on resources with diverse performance and cost characteristics. The management of VMs within each physical machine or server is overseen by a software layer referred to as the hypervisor or VMs monitor, which enables the creation and isolated operation of VMs. In the realm of cloud computing, WS presents a significant challenge, aiming to map workflow activities onto VMs while accounting for a variety of functional and non-functional constraints [26]. Workflows are made up of interdependent activities that are linked together by data or functional requirements, and these relationships must be taken into account when scheduling [27]. However, with cloud computing, WS a computationally demanding issue known as NP-hard optimization, making it difficult to obtain an ideal timetable. The presence of several VMs in a cloud, together with the requirement to plan different user tasks while taking

into account diverse objectives and variables, adds to the complexity. The fundamental goal of WS strategies is to reduce wait time by appropriately distributing jobs to virtual resources [28, 29]. A scheduling method, for example, may be designed to satisfy service level agreements (SLAs), meet user-specified dates, and adhere to cost limits [30]. When making scheduling decisions, scheduling solutions take into account elements such as resource usage, load balancing, and the availability of cloud resources and services [29].

2.2Workflow scheduling objectives

The key WS objective is to accomplish the expected aim by allocating the fitting resources to execute tasks. Currently, the shared WS objective schemes include availability, economic principle, maximum resource utilization, minimum makespan, load balancing, security, and higher dynamic adaptability in the environment of cloud computing. Scheduling strategies are divided into a) probabilistic search, b) heuristics, and c) hybrid approaches. Probabilistic search is an extensive category of scheduling algorithms that include GAs, simulated annealing (SA) [31], and ant colony or swarm optimization. Heuristics are a time-effective resolution for specific problem space scenarios. The most common heuristic is list-based scheduling, which produces a priority list of tasks according to specific standards. Subsequently, tasks are allocated to resources [32]. Scheduling, which involves allocating resources to tasks, is a computational problem that is known to be extremely challenging to solve efficiently. As a result, it is typically addressed using heuristic methods, which are approximate algorithms or strategies that provide practical solutions but may not guarantee an optimal solution. Hybrid approaches combine strategies to augment algorithm performance. For example, [33] hybridized GA with SA, while Daoud and Kharma [34] hybridized heuristics and GA. Furthermore, based on the literature [35] combined ant colony and GA. WS is a crucial concern in workflow execution management. Scheduling plans and directs the implementation of inter-dependent tasks based on distributed resources [36]. Scheduling allocates appropriate resources to workflow tasks to complete execution to fulfill users' aims and objectives and functions. Appropriate scheduling can substantially affect system performance. The fundamental objective of WS is to achieve the intended outcome by appropriately allocating tasks to suitable resources for execution. Presently, widely recognized aims for WS strategies encompass economic considerations, availability, minimized time requirements, optimized resource utilization, security, load distribution, and enhanced dynamic adaptability within the context of cloud computing [19], among various others. The scheduling objectives under scrutiny in this investigation are illustrated in *Figure 2*.

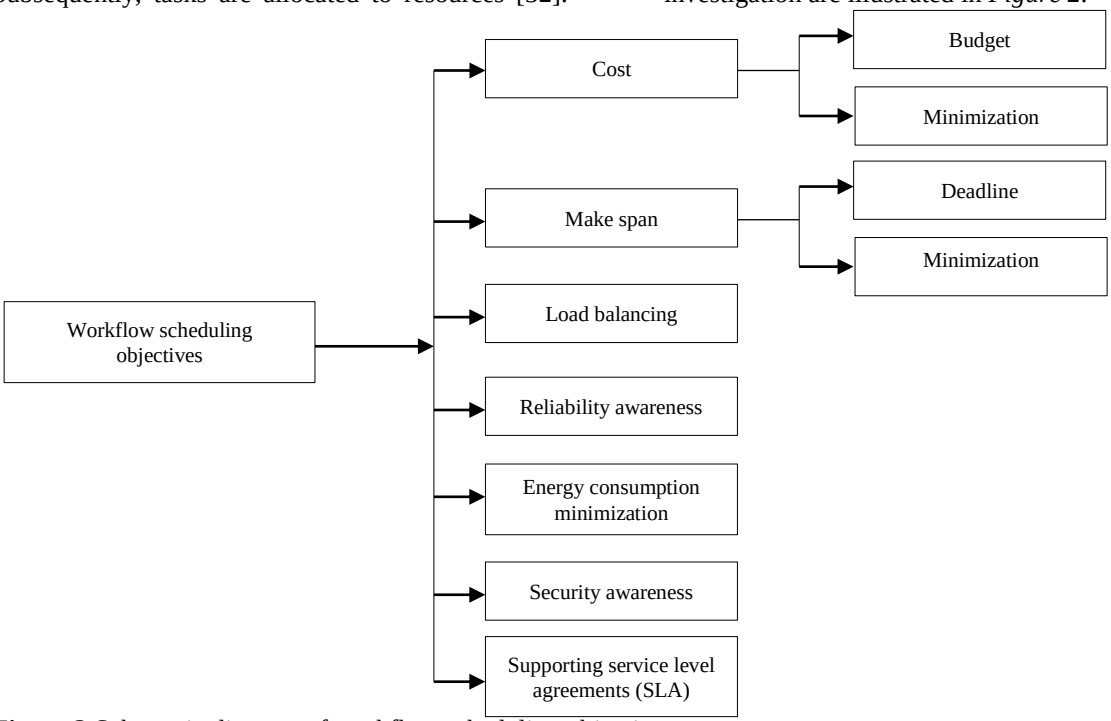

Figure 2 Schematic diagram of workflow scheduling objectives

- **Cost:** The distributed computing nodes within the cloud cluster could be geographically dispersed, necessitating equitable administrative expenses for the cloud client towards the cloud provider. The overall expenditure linked with the execution of processes in the cloud encompasses various cost elements, including but not limited to computational expenses and expenses related to data transmission. These specific cost factors will be explored in greater detail in the subsequent sections. The management of process execution costs has emerged as a significant objective within the realm of cloud WS investigations [37].
- **Makespan:** The timeframe allocated for the execution of a workflow primarily hinges on the combined execution durations of individual tasks and the associated communication expenses between them. In simpler terms, it denotes the span commencing with the initiation of the initial task and concluding with the finalization of the last task. Ever since the inception of cloud computing, attaining the prescribed deadline for a workflow has remained a predominant objective across various scheduling methodologies.
- In the realm of cloud computing, particularly for extensive data processing tasks, maintaining balanced work distribution holds significant significance. A WS approach must encompass the equitable distribution of workloads across diverse nodes within a geographically dispersed and varied environment, characteristic of the cloud. This strategic optimization contributes to enhancing resource utilization efficiency while mitigating the risk of overwhelming any specific resource.
- **Reliability awareness:** Reliability consciousness stands as another crucial prerequisite in the domain of process scheduling. Beyond time and cost considerations, the trustworthiness of workflow execution is taken into account. This aspect signifies the probability of accomplishing a task successfully while adhering to the user's predefined QoS constraints, even in scenarios involving failures of resources or tasks. To enhance this reliability, scheduling algorithms can employ strategies like active replication and backup/restart mechanisms. Nonetheless, these algorithms must diligently evaluate the expenses incurred due to task reiteration, encompassing factors such as time and computational resources squandered[38].
- The pursuit of diminishing energy consumption is rapidly gaining prominence within the realm of cloud computing. With the escalation in demand for cloud services, data centers are consuming significant amounts of energy, underscoring the imperative for enhanced energy efficiency. Cloud service providers are facing mounting pressure to curtail their energy consumption rates. To address this challenge, contemporary algorithms have been devised to navigate the intricacies of harmonizing energy consumption, performance, and expenses. Nonetheless, it's crucial to acknowledge that energy optimization has yet to attain relevance at the VMs abstraction level. In summary, when orchestrating workflows within the context of cloud computing, factors such as load balancing, awareness of dependability, and reduction in energy usage assume heightened significance. These objectives are aligned to optimize the utilization of resources, ensure reliability, and subsequently minimize energy consumption, prioritized in that sequence.

WS is one of the happening research topic in cloud computing. Simulation based approach has become well liked technology to evaluate cloud computing systems, their security and performance and application behaviour's. Several simulators have been particularly developed for evaluation and performance analysis of cloud environments.

2.3Task scheduling algorithms

In the work of Zhong et al. [39] a scheduling approach known as greedy particle swarm optimization (G&PSO) was introduced. The outcomes demonstrated enhancements in various aspects of each VM performance, encompassing both global and local search capabilities. Additionally, the method exhibited accelerated convergence rates and contributed to a more consistent distribution of workloads. In the study conducted by Wei et al. [40] a multi-population genetic algorithm (MPGA) was explored for load balancing, aimed at mitigating premature convergence of tasks within cloud systems. The findings illustrated that the MPGA exhibited strong performance in terms of job scheduling, leading to reductions in execution time and associated costs. Lin and Li [41] devised an approach for scheduling tasks within cloud systems utilizing a pre-allocation ant colony optimization (PACO) framework. The proposed technique demonstrated strong efficiency. According to [42] which introduced a three-stage selection process

along with a genetic strategy termed "total-division-total" to complement their genetic approach [42]. Utilizing the CloudSim tool, the outcomes indicated that the enhanced algorithm outperformed a basic GA concerning the duration of task completion. This highlighted the reliability of the improved GA as a viable approach for scheduling jobs within cloud computing. Gupta et al. [43] introduced a meta-heuristic adaptation of the ant colony optimization (ACO) algorithm for the scheduling of tasks within cloud systems, with a focus on two primary objectives. The results demonstrated that the suggested load-balancing ant colony optimization (LB-ACO), when compared to the non-dominated sorting GA II, non-dominated sorting genetic algorithm (NSGA-II), exhibited superior performance in terms of load distribution equilibrium and makespan reduction. NSGA-II is an evolutionary optimization method that efficiently categorizes individuals based on their non-dominated solutions to effectively address multi-objective optimization problems. In response to scheduling challenges, Wei et al. introduced a scheduling approach termed self-adaptive multi-population genetic algorithm (SAMPGA) [44]. The simulations revealed favorable outcomes, indicating that SAMPGA yielded positive results in relation to cost efficiency, job completion duration, and load balancing.

3.Methodology

In this study, the proposed e-HGA tested two levels of fitness, which qualified a chromosome and a gene to be preferred to their other counterpart in a data population. At the chromosome level, parents were randomly selected from the population and crossed over (mated) to produce offspring that were subsequently mutated by inserting random geneLists to maintain population diversity to avoid premature convergence. The fittest chromosome underwent a second round of native hybrid genetic algorithm (HGA), where two parents (geneList) were selected from the fittest chromosome to mate (crossover). Subsequently, the crossover site geneLists were exchanged, thus producing an entirely new individual (offspring). Following this, the offspring were mutated by inserting random genes to maintain population diversity to avoid premature convergence. This experiment was performed with the same data using a conventional HGA. The results were compared to determine whether the proposed e-HGA or conventional HGA performed better. **Chromosome level:** The term chromosome is used to symbolize the answer, which is an integer string made up of a series of substrings [1]. Each substring represents a list of nodes that a bus has visited. For example, the chromosomal coding for a solution with two routes and five pick-up and drop-off stop pairs (Protein Data (PD) pirs) is as follows: $0\text{-}2^{+}\text{-}4^{+}\text{-}2^{-}\text{-}4^{-}\text{-}0\text{-}1^{+}\text{-}1^{-}\text{-}3^{+}\text{-}5^{+}\text{-}5^{-}\text{-}3^{-}\text{-}0$, where 1^{+}, 2^{+}, 3^{+}, 4^{+}, and 5^{+} represent pick-up stops, while 1^{-}, 2^{-}, 3^{-}, 4^{-}, and 5^{-} represent drop-off stops. 0 represents a terminal, which divides the chromosome into multiple routes. The start and finish terminals are determined by the closeness of the nearest terminals to the first and last stops, respectively. In this investigation, the chromosome is created at random.

Hybrid genetic algorithm (e-HGA): The exploration of a stochastic GA is combined with the accurate convergence of a deterministic method in an e-HGA. The new aspect is the way these algorithms are coupled. e-HGA includes all the processes of a traditional GA (e.g., selection, mating, mutating, elitism), but after the mutation procedure, each candidate is optimized locally, as shown in the algorithm flowchart. The combination of these two techniques results in an aggressive optimization process capable of searching the whole variable space while also converging on the precise local optimum for each candidate. In a hybrid GA, natural selection regulates the placement of new candidates, with the best candidate more likely to be put. The ability to obtain global optimum values that normal stochastic algorithms cannot detect is the key advantage. **Elitism**: Defines the highest proportion of p **Crossover:** This represents exchange of genes. Two chromosomes are selected using a selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring). For the GA. VMs crossed over for this process. **Mutation**: Mutation is a process inserting random genes in offspring to maintain the diversity in the population to avoid premature convergence.

3.1Proposed model

The HGA comprises two levels of architecture for task scheduling with enhanced selection (elitism). The main structure contains all key e-HGA components with a few modifications. The e-HGA calculates fitness based on the size of jobs in million instructions (MI) and the computational power of VMs in MI per second (MIPS) [45]. *Figure 3* depicts the proposed e-HGA model, which is a slight modification of the conventional e-HGA process model. The diagram contains two sections, which represent the e-HGA levels combined to create the e-e-HGA. The model began with level 1 population-level e-HGA. Parent initialization sorted the

population to select the top 10 chromosomes to enable crossover (mating) between two parents from the population. Successively, fitness evaluation examined the level of fitness of each chromosome to select the best chromosome. Subsequently, geneList crossover was conducted to create offspring, followed by mutation. Level 2 chromosome-level e-HGA involved parent initialization to sort the population and select the top 10 geneLists to identify two parents from the best-fit chromosome selected in level 1 HGA. Subsequently, fitness evaluation examined the level of fitness of each geneList to select the best geneList. The next stage was the crossover, where Generalists within the parent chromosomes were exchanged to create a completely new geneList (offspring), while the mutation stage involved inserting random geneLists in the offspring to maintain population diversity to avoid premature convergence [46]. The fittest geneLists were selected from the best chromosomes with the fitness function. The improvement introduced by the proposed e-HGA model was described as follows: elitism (selection of the top 10% fittest individuals) was used to calculate individuals' fitness at each level before the best fit was selected for mating [1].

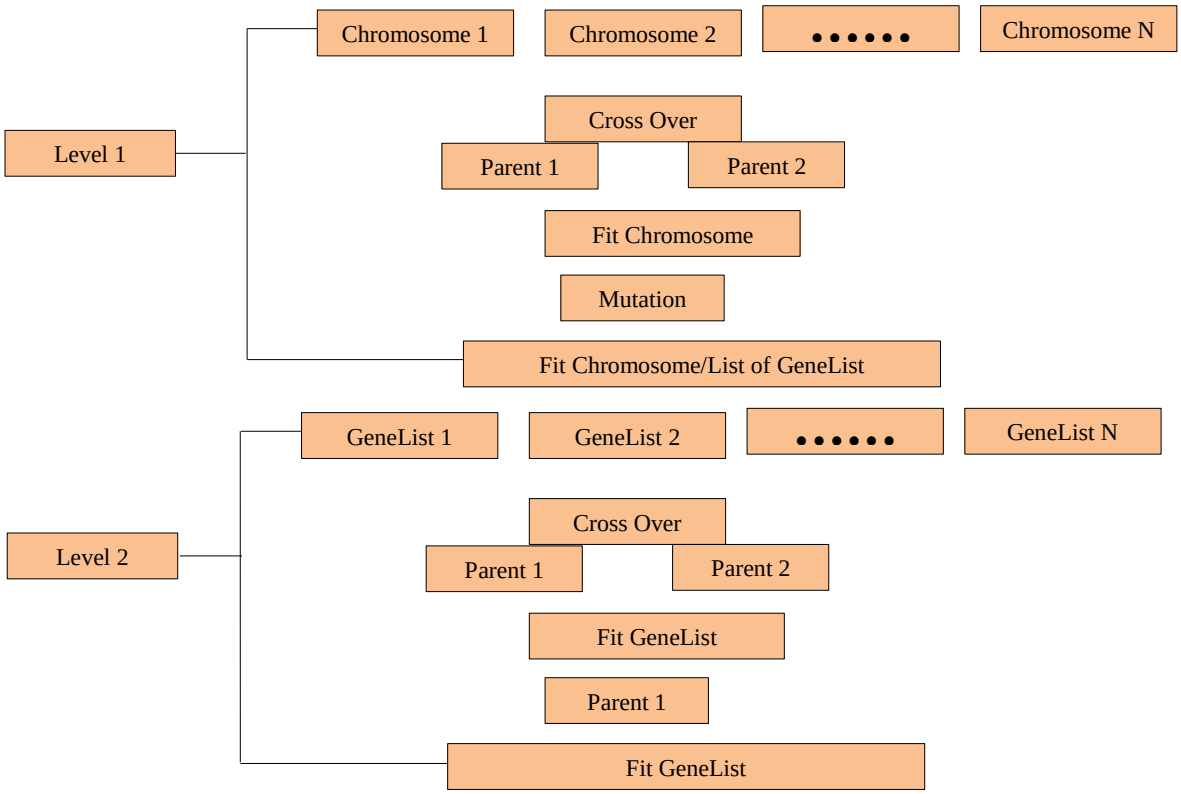

Figure 3 Proposed HGA model

The first-stage population-level e-HGA involved fitness, crossover, mutation, and repeat fitness until convergence. This stage was essential to firsthand fine-tune the geneLists. The subsequent stage was chromosome-level HGA, where the e-HGA processes were repeated. At the crossover stage of both levels, individual exchanges were conducted to create a wholly new individual (offspring) that would proceed to the mutation stage [47]. The new genes inserted during the mutation would be inherited at the next stage (new population). At the decision stage, if the decision outcome was negative, the process was iterated back to the fitness evaluation stage, from which the entire process was repeated until a positive outcome was achieved. Nonetheless, if the decision outcome was positive, the process would proceed to the chromosome fitness evaluation level, where each chromosome was evaluated for best fitness. The process was terminated once the chromosome with the best fitness was found.

The objective function is used to provide a measure of how individuals have performed in the problem domain. In the case of a minimization problem, the fit individuals will have the lowest numerical value of the associated objective function. This raw measure

of fitness is usually only used as an intermediate stage in determining the relative performance of individuals in a e-HGA. Another function, the fitness function, is normally used to transform the objective function value into a measure of relative fitness, where f is the objective function, g transforms the value of the objective function to a non-negative number and F is the resulting relative fitness. This mapping is always necessary when the objective function is to be minimized as the lower objective function values correspond to fitter individuals. In many cases, the fitness function value corresponds to the number of offspring that an individual can expect to produce in the next generation. A commonly used transformation is that of proportional fitness assignment (see, for example, [1]). The individual fitness, *F(xi)*, of each individual is computed as the individual's raw performance, *f(xi)*, relative to the whole population, i.e. (Equation 1),

$$F(x_i) = F(x_i) \; / \; \sum_{x=1}^{Nind} F(xi) \qquad (1)$$

Where N_{ind} is the population size and xi is the phenotypic value of individual i. whilst this fitness assignment ensures that each individual has a probability of reproducing according to its relative fitness, it fails to account for negative objective function values.

A linear transformation which offsets the objective function [1] is often used prior to fitness assignment, such that, where a positive scaling factor if the optimization is maximizing and negative if we are minimizing. The offset b is used to ensure that the resulting fitness values are non-negative.

3.1.1A step-by-step description of the e-HGA algorithm's workflow

The architecture of e-HGA enhances HGA for task scheduling at two levels with enhanced selection process (Elitism). The main structure has all major components of e-HGA with a few modifications. e-HGA requires size of jobs in MI and computational power of VMs in MIPS to calculate fitness. The model starts with level 1 HGA at the population level. Initialization of parents which sorts the population to selects the top 10 Chromosomes to make two parents from the population to mate (Cross Over). The next stage is the evaluation of fitness which examine the fitness level of each Chromosome to select the best Chromosome to be used in the process, the next stage is the crossover which exchanges geneList within the parent Chromosomes to create a completely new Chromosome that can be called an offspring while the mutation stage inserts random geneList in offspring to maintain the diversity in the population to avoid premature convergence.

Level 2 e-HGA at the Chromosome level involves initialization of parents which sorts the population to select the top 10 geneList to make two parents from the best fit Chromosome selected in level 1 e-HGA. The next stage is the evaluation of fitness which examine the fitness level of each geneList to select a best geneList to be used in the process, the next stage is the crossover which exchanges geneList within the parent Chromosomes to create a completely new geneList that can be called an offspring while the mutation stage inserts random geneList in offspring to maintain the diversity in the population to avoid premature convergence. The fitness function is further used to pick the fittest GeneList from the best Chromosomes. These levels of e-HGA are demonstrated in *Figure 4*.

The following operation explains the improvement introduced by HGA model.

1. Using Elitism which is the selection of the top 10 percent fit individuals to calculate fitness of individual at each level before picking the best fit for mating (cross over).
2. Do the native HGA at two levels. Make decisions on population using native e-HGA methods fitness, crossover and mutation.

Figure 4 represents the proposed e-HGA process model which is a slight modification of the conventional HGA process model. The diagram is sectioned into two levels, which represent the different levels of HGA combined to create the e-HGA. The first stage is HGA at population level. At this level, the process of HGA is done which is fitness, cross over, and mutation and repeat Fitness until convergence. This stage is essential to firsthand fine-tune geneLists which are children of a Chromosome. The next stage is HGA at chromosome level. At this level, the process of HGA is repeated which is fitness, cross over, and mutation and repeat Fitness until convergence. At the stage of crossover in both levels, an exchange of individuals is done to create a completely new individual that can be called an offspring that will be passed on to the next mutation stage. The mutation process inserts random genes in individuals to maintain the diversity in the population so as to avoid premature convergence. The new genes are now passed on to the next stage called new population. At the decision stage of the process, if the outcome of the decision is negative the process iterates back to the evaluation of fitness stage

and from there the whole process is repeated until the outcome is positive. But on the other hand, if the decision outcome is positive then the process proceeds to the evaluation of fitness of chromosome level, at this stage each chromosome is evaluated for best fitness and once the chromosome with the best fitness is found the process terminates.

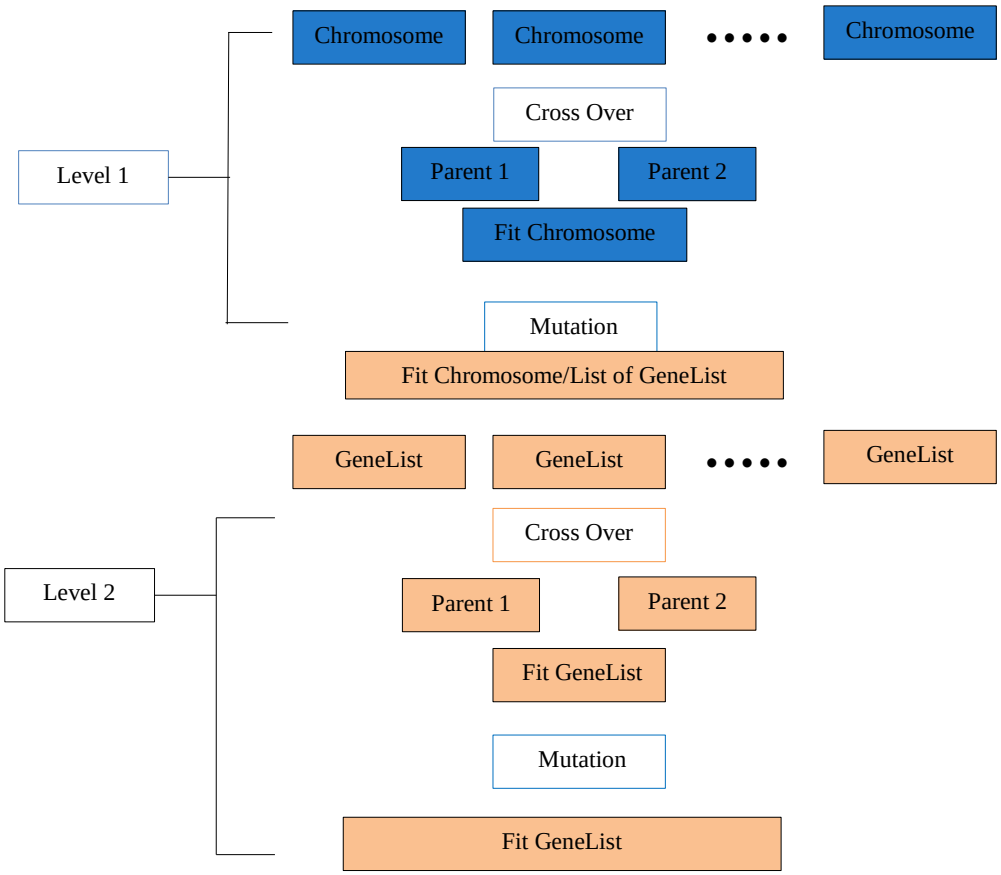

Figure 4 Proposed e-HGA Model

Figure 5 shows the overall pseudo code for e-HGA. Line 1 sorts of the unsorted list of VMs and places it in a place holder called vmList, line 2 also sort the unsorted cloudlets and place it a place holder called cloudletList, line 3 randomizes the number of cloudlets form 0 with the number of VMs and place them in a place holder called population. Line 4 performs a doCrossOver iteration and a do Mutation iteration on the cloudlets placed in the population placeholder until a BestFitness is found, line 5 closes the iteration, line 6 initializes the best population to become the modified population, line 7 performs a loop were best Chromosomes in the population equals to fittest chromosomes found in the best population of chromosomes, line 8 close the whole algorithm. *Figure 6* shows a (while) iteration operations performed to find the best population, line 1 initializes foundBestFit to be false, line 2 Initializes previous fittest to be minimum Integer, line 3 initializes fitness to equal 0. Line 4 performs a while loop operation were found best fit is not false as basic parameter, at the end of the (while) operation, the last line of the general code returns a best population.

```
vmList ← Sort (unsortedVmList)
cloutletList ←Sort (unsortedCloudletlist)
population ← Random (from 0 until Number of Cloudlets ×Number of VMs)
Until (bestFitness) { // Iteration Algorithm
 doCrossOver (population)
doMutation (population)
}
bestPopulation = modified population
for (each Chromosome in bestPopulation) {
        bestChromosome = findFitestChromosome ()
}
```

Figure 5 Illustration of e-HGA unsorted list of VM

```
foundBestFit = false
 previousFitest = MinimumInteger
 fitness = 0
while (foundBestFit == false) {
            tempFitValue = 0
            for i ←0 to populationSize{
             fit = fitnessFunction
             tempFitValue += fit
            }
            fitness = tempFitValue
             if fitness < previousFitest:
             foundBestFit = true
             if foundBestFit{
            return bestPopulation
            end while Loop
            }
            previousFitest = fitness
            doCrossOver:
            doMutation ()
}
return bestPopulation
```

Figure 6 Iteration operations performed

Figure 7 shows the cross over iteration on the population size of chromosomes. Line 1 assigns all the results of the operation performed on the population size to (i) in the four loops. Line 2 randomizes index number 1 in the loop and assigns it to variable x. Line 3 randomizes index number 2 in the loop and assigns it to variable y, line 4 getGeneList is multiplied with variable x and is initialized to 1. Line 5 getGeneList is multiplied with variable y and is initialized to 2, line 6 randomize new index number 1 in the loop and assigns it to variable I, line 7 randomize new index number 2 in the loop and assigns it to variable j. Line 8 gets VM from list 1 and in variable i and assigns it to Vm1, line 9 gets VM from list 1 and in variable j and assigns it to Vm2. Line 10 updates the chromosome gene with Vm1 at variable I, line 11 updates the chromosome gene with Vm2 at variable j. *Figure 8* shows the mutation probability iteration done on the chromosome population. Line 1 randomizes an empty parameter and assigns it to mutation probability (mutProb). Line 2 performs if conditions were the below lines are executed if the mutProb is less than 0.5. Line 3 initializes a random to randomNumber, line 4 gets the GeneList and random Number and merges it with the population and all is assigned to mutGeneList. Line 5 assigns mutGeneList of the Chromosomes to mutChromosome, line 6 randomizes an empty parameter and assigns it to randomIndex. Line 7 randomizes an empty parameter and assigns it to randomIndex2, line 8 gets the randomIndex1 and inculcates it with VmList and assigns them to mutVM. Line 9 updates mutVM in random Index 2, and inculcates them with mutChromosome, Line 10 sets population randomNumber. Line 11 closes the entire conditional if statement.

```
for i ←  populationSize {
            x = randomlndex1
             y = random Index2
             I1 ← getGeneList (x)
            I2 ← getGeneList (y)
             i = randomlndex1
             j = randomIndex2
             Vm1← getVmFromLl (i)
            Vm2 ← getVmFromL1 (j)
chromosome.updateGene(with Vml at i)
 chromosome.updateGene(with Vm2 at j)
}
```

Figure 7 Shows the cross over iteration on the population size of chromosomes

```
mutProb = Random ()
if mutProb < 0.5{
            randomNumber = Random ()
 mutGeneList =
population.get(randomNumber).getGenelist()
 mutChromosome = Chromosome (mutGeneList)
randomlndex1 = Random ()
 randomIndex2 = Random ()
 mutVM = VmList.get (randomlndex1)
 mutChromosome.update (mutVM at randomIndex2)
population.set (randomNumber)
}
```

Figure 8 Shows the mutation probability iteration done on the chromosome population

Figure 9 shows the code for finding the fittest chromosomes in the population of chromosomes. Line 1 is an if statement condition were the population size is assigned to variable i. Line 2 gets the list of gene at I and assigned the list to GeneList. Line 3 calculates the fitness of the GeneList, Line 4 closes the if statement. Lines 5 get the fittest geneList and add it up to the population and the result is said to be the bestChromosome.

```
for i ←populationSize{
            GeneList ← population.getList (gene at i)
            calculateFitness ()
}  bestChromosome ←population.get (fittest geneList)
```

Figure 9 The Code fittest chromosomes in the population of chromosomes

3.2Simulation tool

The project data was simulated with the Eclipse tool. Eclipse simulates data more rapidly without compromising results and requiring hardware upgrades. An improved constrained pressure residual (CPR) solver enables additional residual checks for marked performance improvements, specifically when repeated non-linear convergence issues occur. Eclipse simulates faster without compromising results and without upgrading your hardware. Improved CPR solver enables you to provide addition residual checks, for dramatic improvements in performance, especially when there are repeated non-linear convergence problems.

3.3Test environment

CloudSim comprises a simulation engine, cloud services, and source code, positioning itself as an extensible provisioning environment. Its layered construction mirrors the hierarchical nature of cloud computing environments. Given the rapidly evolving nature of cloud computing as a research area, there exists a shortage of well-defined standards, tools, and methods capable of effectively addressing the complexities at both the infrastructure and application levels. *Figure 10* illustrates the layered structure of the CloudSim software framework and its architectural components. The bottom layer features the SimJava discrete event simulation engine [48], which serves as the foundation by implementing essential functionalities for higher-level simulation frameworks. These functionalities encompass event queuing and processing, the creation of system components (services, host, data center, broker, and VM component communication, and simulation clock management. Moving up the layers, the next tier consists of libraries that implement the GridSim toolkit [49]. This toolkit provides support for high-level software components that model various Grid infrastructures, including networks and associated traffic profiles. Additionally, it includes fundamental Grid components such as resources, data sets, workload traces, and information services. The CloudSim layer is situated above the GridSim layer and is implemented by programmatically extending the core functionalities offered by GridSim. CloudSim introduces innovative support for modeling and simulating virtualized Cloud-based data center environments, featuring dedicated management interfaces for VMs, memory, storage, and bandwidth.

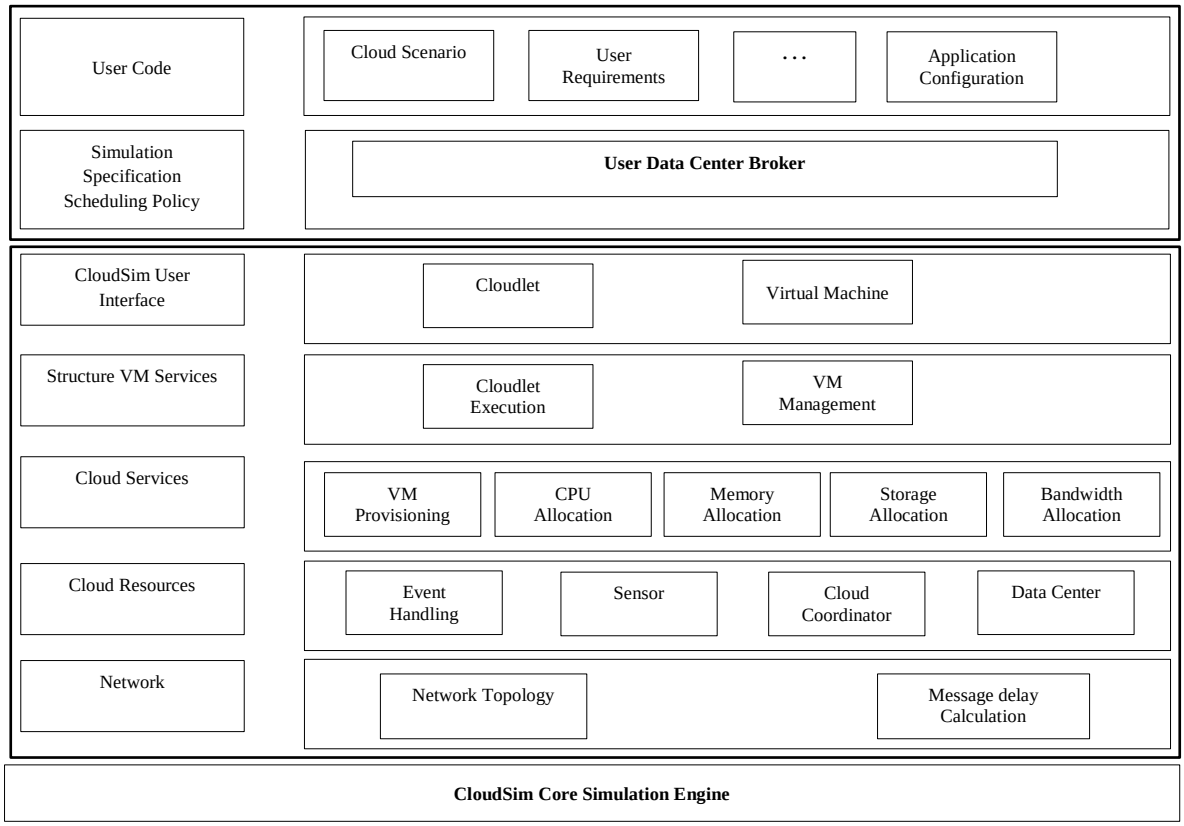

Figure 10 The layered structure of the CloudSim software framework and its architectural components

In this layer, cloudSim is responsible for managing the instantiation and execution of core entities such as VMs, hosts, data centers, and applications throughout the simulation period. It possesses the capability to concurrently instantiate and seamlessly manage a large-scale cloud infrastructure comprising thousands of system components. Essential tasks like provisioning hosts to VMs based on user requests, managing application execution, and dynamic monitoring are efficiently handled by the cloudSim layer. For a cloud provider seeking to investigate the effectiveness of different policies in host allocation, strategic implementations need to be incorporated at this layer by programmatically extending the core VM provisioning functionality. A notable distinction at this layer is the methodology employed in allocating a host to various competing VMs in the cloud. A cloud host can be concurrently shared among multiple VMs executing applications based on user defined QoS specifications. The proposed e-HGA step by step is presented below:

Step 1: e-HGA at initialization level.

```
Initialize and assign cloudLets to VM.
Input: CLoudletList, VmList
        averageNumber = cloudletsSize / VmSize;
int rem = cloudletsSize % VmSize
        int noVms = vmlistSize;
        // Assign Cloudlets to VM
            population = new List();
        fittnessList = new ArrayList ();

        for(int x = 0; x < 100; x++) {
        chromosome = new List();
        for(int j = 0; j < 20; j++) {
        Shuffle(vmlist);
        Shuffle(Cloudletslist);
        oldPosition = 0;
        genelist = new List<>();
        for(int i = 0; i < noVms; i++) {
 if(i == noVms - 1 && rem != 0) { // Assigning  the
remaining Cloudlets to Vm if any exist.
geneLast = new Gene2(cloudletList[oldPosition:
cloudletListSize], vmlist[i]);
genelist.add(geneLast);
break;
}
clts = cloudletList[oldPosition : (i + 1)×aveCls];
gene = new Gene2(clts, vmlist[i]);
oldPosition = (i + 1) × averageNumber;
 if(i < noVms) {
        genelist.add(gene);
}
}
chromosome.add(genelist);
}
population.add(chromosome);
}
```

Step 2: e-HGA at Population Level.

```
Input: CLoudletList, VmList, Population
converged = false;
convValues = new List();
convValues.add(average(fitnessPopulation(populatio
n)));
best = population //From initialization
while(converged == false) {
srt = sortPopulation(population);
        newGeneration = new List ();
// ELITISM: 10% of the best
newGeneration = srt[0, (10 × srt.size/100)];
parent1 = new List();
parent2 = new List();
//CROSS_OVER or MUTATION
parent1.addAll(srt[0, srt.size]);
parent2.addAll(srt[0, srt.size]);
p = mate(parent1, parent2, cloudletList, vmlist);
        newGeneration.addAll(p);
population = newGeneration;
avg = average(fitnessPopulation(population));
miniValue = Collections.min(convValues)
if(avg < miniValue) {
best = newGeneration;
}
sort(convValues);
if(convValues.size > 500 || avg < min(convValues ))
{
converged = true;
}
if(convValues.size() > 500) {
converged = true;
}
convValues.add(avg);
}
bestChromosome = pickBestChromosome(best);
```

Step 3: e-HGA at chromosome level.

```
Input: CLoudletList, VmList, bestChromosome.
bestGeneList = population[0]; // INITIALISED
bestGeneList from one chromosome, the value will
be reassigned
converged = false;
while(converged == false) {
        srt = sortChromosome(bestChromosome);
newGeneration = new List();
// ELITISM: 10% of the best
newGeneration = srt[0, (10 × srt.size/100)];
parent1 = new List();
parent2 = new List();
//CROSS_OVER or MUTATION
```

```
parent1.addAll(srt[0, srt.size]);
parent2.addAll(srt[0, srt.size]);
p = mateChromosomes(parent1, parent2,
cloudletList, vmlist);
newGeneration.addAll(p);
avg = fitnessChromosome(newGeneration);
minimum = Collections.min(convValues)
if(avg < minimum) {
bestGeneList = newGeneration;
}
sort(convValues);
minimum = Collections.min(convValues)
if(convValues.size() > 500 || avg < minimum {
converged = true;
}
if(convValues.size() > 500) {
converged = true;
}
convValues.add(avg);
}
```

Step 4: Final assignment and running cloudsim

```
Input: CLoudletList, VmList, bestGeneList.
bestGeneListList = pickBestGenelist(bestGeneList);
clts = new List();
for(Gene2 gene: bestGeneListList) {
for(i = 0; i < gene.getCloudletsFromGene().size();
i++) {
clt = gene.getCloudletsFromGene().get(i);
clts.add(clt);
clt.setUserId(brokerId);
clt.setVmId(gene.getVmFromGene().getId());
}
}
broker.submitCloudletList(clts);
broker.submitVmList(vmlist);
Fitness Population
Input: Population.
function fitnessPopulation {
fitness = 0;
fitnessList = new List();
for(List<Gene2> chromo : population) {
fitness = fitnessChromosome(chromo);
fitnessList.add(fitness);
}
return fitnessList;
}
Fitness Chromosome
function fitnessChromosome(List<List<Gene2>>
chromo) {
fitness = 0;
for (List<Gene2> cr1: chromo) {
for (Gene2 gene: cr1) {
        fitness +=  fitnessGene(gene);
}
}
return fitness;
}
Fitness Gene
function fitnessGene(Gene2 gene) {
int fitness = 0;
Vm vm = gene.getVmFromGene();
List<Cloudlet> clt = gene.getCloudletsFromGene();
for (Cloudlet ct: clt) {
fitness += ct.getCloudletLength() / vm.getMips();
}
return fitness;
}
PickBestGeneList function:
function pickBestGenelist(List<List<Gene2>>
chromosome){
best= new List();
bestValue = Integer.MAX_VALUE;
for(int i = 0; i< chromosome.size; i++)
{
int fitValue =
fitnessChromosome2(chromosome.get(i));
if(fitValue < bestValue)
{
best = chromosome.get(i);
bestValue =
fitnessChromosome2(chromosome.get(i));
}
}
return best;
}
Mate function:
Input:
static mate(List<List<List<Gene2>>> parent1,
List<List<List<Gene2>>> parent2, sortedList,
List<Vm> sortedListVm){
children = new List();
for(int x = 0; x < (90 × parent1.size/100); x++) {
Random rand = new Random();
Double randNum = rand.nextDouble(1);
if(randNum < 0.45) {
children.add(parent1.get(x));
} else if(randNum <= 1.0) {
        children.add(parent2.get(x));
}
}
return children;
}
Mate Chromosomes:
static mateChromosomes(List<List<Gene2>>
parent1, List<List<Gene2>> parent2, List<Cloudlet>
sortedList, List<Vm> sortedListVm){
        = new List();
for(int x = 0; x < (90 × parent1.size()/100); x++) {
```

```
Random rand = new Random();
Double randNum = rand.nextDouble(1);
if(randNum < 0.45) {
        children.add(parent1.get(x));
} else if(randNum <= 1.0) {
        children.add(parent2.get(x));
}
}
return children;
}
Fitness Chromosome 2
static int fitnessChromosome2(List<Gene2> chromo)
{
        int fitness = 0;
for (Gene2 geneList: chromo) {
fitness +=  fitnessGene(geneList);
}
return fitness;
}
// Explanation of HGA Mechanism
e-HGA at Population Level.
Input: CLoudletList, VmList, Population
converged = false;
convValues = new List();
        convValues.add(average(fitnessPopulation(p
opulation)));
best = population //From initialization
while(converged == false) {
srt = sortPopulation(population); // Sorts population
according to Cloudlet length / MIPS
newGeneration = new List();
// ELITISM: 10% of the best
newGeneration = srt[0, (10 × srt.size/100)]; //Pick the
Top 10% Best
parent1 = new List(); // Initialize one parent
parent2 = new List(); // Initialize another parent
//CROSS_OVER or MUTATION
parent1.addAll(srt[0, srt.size]); // Adds all Top 10%
Best to one parent
parent2.addAll(srt[0, srt.size]); // Adds all Top 10%
Best to another parent
p = mate(parent1, parent2, cloudletList, vmlist); //
Mate parents, mutation and cross over randomly.
newGeneration.addAll(p); // Adds all to a
allGeneration.
population = newGeneration;
avg = average(fitnessPopulation(population)); //
Calculate the average of all population fitness.
miniValue = Collections.min(convValues) // Finds
the minimum of averages to determine convergence.
if(avg < miniValue) { // Determines convergence.
best = newGeneration; // If converged, save the best
}
sort(convValues); // Sort the average values to be
used in the loop.
if(convValues.size > 500 || avg < min(convValues ))
{ // Terminate loop if not converged at 500 iteration
converged = true;
}
if(convValues.size() > 500) {// Terminate loop if not
converged at 500 iteration
converged = true;
}
convValues.add(avg); //Adds average value to be
tested for convergence.
}
bestChromosome = pickBestChromosome(best); //
Pick best chromosome.
e-HGA at Chromosome Level.
Input: CLoudletList, VmList, bestChromosome.
bestGeneList = population[0]; // INITIALISED
bestGeneList from one chromosome, the value will
be reassigned
converged = false;
while(converged == false) {
srt = sortChromosome(bestChromosome); // Sorts
population according to Cloudlet length / MIPS
newGeneration = new List();
// ELITISM: 10% of the best
newGeneration = srt[0, (10 × srt.size/100)]; //Pick the
Top 10% Best
parent1 = new List(); // Initialize one parent
parent2 = new List(); // Initialize another parent
//CROSS_OVER or MUTATION
parent1.addAll(srt[0, srt.size]); // Adds all Top 10%
Best to one parent
parent2.addAll(srt[0, srt.size]); // Adds all Top 10%
Best to another parent
p = mateChromosomes(parent1, parent2,
cloudletList, vmlist); // Mate parents, mutation and
cross over randomly.
newGeneration.addAll(p); // Adds all to a
allGeneration.
avg = fitnessChromosome(newGeneration); //
Calculate the fittest chromosome.
minimum = Collections.min(convValues) // Finds the
minimum of averages to determine convergence.
if(avg < minimum) { // Determines convergence.
bestGeneList = newGeneration; // If converged, save
the bestGeneList
}
sort(convValues); // Sort the average values to be
used in the loop.
minimum = Collections.min(convValues)
if(convValues.size() > 500 || avg < minimum { //
Terminate loop if not converged at 500 iteration
converged = true;
```

```
}
if(convValues.size() > 500) {// Terminate loop if not converged at 500 iteration
converged = true;
}
convValues.add(avg); //Adds average value to be tested for convergence.
}
Final assignment and running Cloudsim
Input: CLoudletList, VmList, bestGeneList.
bestGeneListList = pickBestGenelist(bestGeneList);
clts = new List();
for(Gene2 gene: bestGeneListList) {
for(i = 0; i < gene.getCloudletsFromGene().size();
i++) {
clt = gene.getCloudletsFromGene().get(i); // Gets CloudLet
        clts.add(clt);
clt.setUserId(brokerId); Set BrokerId of CloudLet
        clt.setVmId(gene.getVmFromGene().getId()
); // Set VmId of Cloudlet to bind it to the VM
}
}
broker.submitCloudletList(clts);
         broker.submitVmList(vmlist);
```

The assessment is conducted based on the fitness function outlined in Equation 1. Elitism is introduced in e-HGA at lines 9 and 10, ensuring the replication of the best chromosomes from each generation to the next. This inclusion prevents the deterioration of solution quality in e-HGA. The selection process at line 19 is detailed in step 2, where fit chromosomes are chosen using binary tournament selection for genetic operations. The selected chromosomes then undergo modified crossover and mutation processes, distinguishing the proposed e-HGA from existing HGAs. Notably, both genetic operators' effectiveness is doubled through the combination of single- and double-point crossover and mutation. The pseudocode for these crossover and mutation routines is presented in steps 3 and 4, respectively. Following each generation, a neighborhood search is executed using a load-balancing function, as outlined in algorithm 5. The conclusion of each generation is marked at line 28 of step 1, and the algorithm iterates to line 5 for the next generation until termination criteria are satisfied.

4. Results

The primary distinguishing factor in cloud WS as opposed to scheduling in multiprocessor or grid systems, is the emphasis on utility. In the latter two systems, scheduling primarily revolves around meeting deadlines or minimizing the makespan of a workflow. In contrast, in the "pay-as-you-go" cloud environment, economic cost is as crucial as performance. Resources are typically provisioned as VMs in the cloud, with virtualization technology forming the foundation of IaaS clouds. This technology is a key differentiator from utility grid computing [50]. However, current virtualization technology falls short of providing stable performance guarantees. Sharifi et al. [51] documented an overall central processing unit (CPU) performance variability of 24% on Amazon's EC2 cloud, attributing it to resource sharing and competition among co-scheduled VMs on the same physical machine. This performance instability renders scheduling methods reliant on task runtime estimations unsuitable in such an environment. Consequently, new WS Methods have been developed specifically for cloud environments.

WS is a well-known NP hard problem that is being studied extensively by researchers to enhance workflow execution performance. NP-hard problems are generally referred to as problems that can be reduced to a different problem that is solved using a polynomial time on a nondeterministic machine, such as an optimization problem, a fractional knapsack, a travel salesman, etc. The experimental performance matrix of our e-HGA as against a regular HGA using some variance of CloudLet while the experiment was done using an increasing number of VM, the result of our experiment shows that the e-HGA performed better than a regular HGA The performance metric used is makeSpan. MakeSpan is the maximum time it takes for a single VM to complete all tasks or jobs assigned to it. In our experiment, for example, in *Table 1*, 15 cloudlet tasks were assigned to a VM. The makespan time of a conventional HGA took 9.668 seconds, while that of an e-HGA took 8.996 seconds. These show that the e-HGA performs better than a normal HGA. The graph in Figure 11 shows that the blue bars, representing the HGA, are taller than the red bars, which represent the e-HGA. The ascending order of the graph shows that the outcome of the experiment increases as the number of cloudlets increases. This study considered cloudlet characteristics with different lengths with a constant network environment and VM with different MIPS. In addition, the e-HGA algorithm was tested with varying VMs and cloudlets (Tasks) to observe their behavior. The same process was repeated for: shortest job first and first come first serve algorithm system. The algorithms' performance was recorded based on the performance metrics. It is recommended

that further research work should be carried out in the area of workflow scheduler using different type HGA to further optimize the workflow result in the area of efficiency and resource management.

The primary distinguishing factor between WS in the cloud and in multiprocessor or grid systems lies in the emphasis on utility in the latter two systems, scheduling revolves around meeting deadlines or minimizing the overall time required for a workflow. However, in the cloud environment, where users are charged for resources based on usage, economic costs are equally important as performance considerations [52]. Cloud resources are typically provided as VMs, and the underlying virtualization technology serves as the foundation for constructing IaaS clouds. This technology is crucial and sets cloud computing apart from utility grid computing. In the context of this study, a population refers to a collection of simultaneous search points or sets of chromosomes (or individuals). Each iterative step that produces a new population is referred to as a generation. The HGA is a GA that has been combined with a local search procedure. The hybridization of GA with a gradient-based search method can help overcome certain limitations specific to GA. With each iteration, this hybridization can enhance the exploration of the solution search space, ultimately reducing computation time. To evaluate the performance of the proposed e-HGA compared to a conventional HGA, an experiment was conducted using an increasing number of VMs [53]. The experimental performance metrics of both algorithms are presented in the Tables and graphs in the subsequent sections, specifically focusing on cloudlet variance.

Table 1 shows the result of the experiment using e-HGA in comparison to HGA. From *Table 1*, makeSpan is the time taken for a task (Cloudlet) to complete using the HGA against e-HGA. The experiment started using 15 cloudlets assigned to 5 VM and our cloudlets were increased by 5 in each experiment. It takes the HGA 9.668 seconds to complete the task, while our e-HGA takes 8.996 seconds to execute the same task. The experimental performance matrix of the e-HGA as against a regular HGA using some variance of cloudLet while the experiment was done using an increasing number of VM, the result revealed that the e-HGA performed better than a regular HGA. The performance metric used is makeSpan. MakeSpan is the maximum time it takes for a single VM to complete all tasks or jobs assigned to it. In our experiment, for example, in *Table 1*, 15 cloudlet tasks were assigned to a VM. The make-span time of a conventional GA took 9.668 seconds, while that of a HGA took 8.996 seconds. These show that the e-HGA performs better than a normal HGA.

Table 1 5 VM MakeSpan of HGA against e-HGA

	5 Virtual machines		
		MakeSpan	
S/N	**Cloudlets**	**HGA**	**e-HGA**
1	15	9.668	8.996
2	20	11.534	9.172
3	25	12.034	11.712
4	30	13.65	13.18
5	35	14.568	14.9
6	40	16.204	16.566
7	45	17.82	17.886
8	50	20.362	19.542
9	55	21.18	21.17
10	60	22.59	21.18

Table 2 presents the experimental findings related to the performance of the conventional HGA and the proposed e-HGA. The evaluation metric used in this study was the makeSpan, which represents the time required for task completion (cloudlet) using either algorithm. The makeSpan serves as a valuable metric for assessing the performance of VM executing cloudlets in cloud computing. It quantifies the total time required for completing a set of tasks, making it a crucial indicator of scheduling efficiency. A reduced makeSpan signifies effective resource allocation, optimized task execution, and minimized idle time on VMs thereby indicating a more efficient scheduling strategy in the context of cloud computing. The experiment commenced with 20 cloudlets assigned to 10 VMs. The conventional HGA completed the task in 9.014 s, whereas the e-HGA accomplished it in 8.698 s. With 30 cloudlets

on 10 VMs, the conventional HGA and e-HGA required 9.95 s and 9.148 s, respectively, for task completion. Increasing the number of cloudlets to 40, the conventional HGA and e-HGA took 10.674 s and 9.558 s, respectively. When 50 cloudlets were assigned to 10 VMs, the conventional HGA completed the task in 11.01 s, while the e-HGA required 12.863 s. Subsequently, with 60 cloudlets on 10 VMs, the conventional HGA and e-HGA achieved task completion in 14.74 s and 14.242 s, respectively. For 70 cloudlets on 10 VMs, the conventional HGA and e-HGA required 15.38 s and 17.25 s, respectively. Based on the research conducted by [54] that used the same method to decision-making to cut server-side computation time and cost. To optimize the VM both locally and globally, this study presents hybrid optimization. The parameter-efficient fine-tuning (PEFT) algorithm was used to initialize the system and functioned as a heuristic algorithm. This approach lowers the error associated with random optimization initialization. Flower pollination with grey wolf optimization (GWO) utilizing a hybrid technique yields significantly better end results than flower pollination with a GA. The suggested method additionally took into account the dependability parameter for various operations [54].

Table 2 The 10-VM MakeSpan of e-HGA against a conventional HGA

10 Virtual machines			
S/N		**MakeSpan**	
	Cloudlets	**HGA**	**e-HGA**
1	20	9.014	8.698
2	30	9.95	9.148
3	40	10.674	9.558
4	50	11.01	12.863
5	60	14.74	14.242
6	70	15.38	17.25
7	80	16.56	17.06
8	90	19.2	18.84
9	100	19.5	18.53
10	110	21	20.72

Moving on to 80 cloudlets on 10 VMs, the conventional HGA and e-HGA took 16.56 s and 17.06 s, respectively, for task completion. With 90 cloudlets on 10 VMs, the conventional HGA and e-HGA required 19.2 s and 18.84 s, respectively. Similarly, for 100 cloudlets on 10 VMs, the conventional HGA and e-HGA achieved task completion in 19.5 s and 18.53 s, respectively. Finally, with 110 cloudlets on 10 VMs, the conventional HGA and e-HGA completed the task in 21.0 s and 20.72 s, respectively. The experimental results indicate that the proposed e-HGA generally outperformed the conventional HGA in terms of task execution time. However, it is noteworthy that the conventional HGA exhibited faster performance than the e-HGA when 50 and 70 cloudlets were assigned to 10 VMs. Thus, further research is necessary to elucidate these findings more comprehensively.

Figure 11 illustrates the graph for the 50 and 70 cloudlets experiment; the blue conventional HGA bars are all taller than the red e-HGA bars. The ascending order of the graph demonstrates that the experiment outcome increased together with the number of cloudlets. *Table 3* presents the experimental results comparing the performance of the conventional HGA with the proposed e-HGA. The experiment involved increasing the number of cloudlets by 20 in each iteration starting with 40 cloudlets assigned to 20 VMs. starting with 40 cloudlets assigned to 20 VMs. The conventional HGA completed the task in 10.1 s, while the proposed e-HGA achieved task completion in 9.77 s for this initial setup. As 60 cloudlets were executed on 20 VMs, the conventional HGA and e-HGA required 10.9 s and 10.1 s, respectively, for task completion. With 80 cloudlets on 20 VMs, the conventional HGA and e-HGA took 13.97 s and 10.47 s, respectively, to complete the task.

Table 3 The 20-VM MakeSpan of enhanced e-HGA against a conventional HGA

20 Virtual machines			
S/N		**MakeSpan**	
	Cloudlets	**HGA**	**e-HGA**
1	40	10.1	9.77
2	60	10.9	10.1
3	80	13.97	10.47
4	100	15.07	14.53
5	120	16.6	15.57
6	140	18.98	17.38
7	160	20.8	18.93
8	180	21.94	19.18
9	200	24.57	20.85
10	220	26.51	22.43

Increasing the number of cloudlets to 100 on 20 VMs, the conventional HGA and e-HGA completed the task in 15.07 s and 14.53 s, respectively. For 120 cloudlets on 20 VMs, the conventional HGA and e-HGA achieved task completion in 16.6 s and 15.57 s, respectively. Similarly, with 140 cloudlets on 20 VMs, the conventional HGA and e-HGA required 18.98 s and 17.38 s, respectively, for task completion. With 160 cloudlets on 20 VMs, the conventional HGA and e-HGA completed the task in 20.8 s and 18.93 s, respectively. Moving on to 180 cloudlets on

20 VMs, the conventional HGA and e-HGA required 21.94 s and 19.18 s, respectively, for task completion. With 200 cloudlets on 20 VMs, the conventional HGA and e-HGA achieved task completion in 24.57 s and 20.85 s, respectively. Furthermore, with 220 cloudlets on 20 VMs, the conventional HGA and e-HGA completed the task in 26.51 s and 22.43 s, respectively. Consistent with previous findings, the results clearly demonstrate that the proposed e-HGA outperformed the conventional HGA in terms of task execution time. *Table 4* depicts the experimental results of increasing the cloudlet number by 30 for each experiment. The experiment began with 60 cloudlets assigned to 30 VMs. The conventional HGA required 11.45 s to complete the task while the proposed e-HGA required 9.074 s to execute the same task. As soon as 90 cloudlets were executed on 30 VMs, the conventional HGA and e-HGA achieved task completion in 14.49 s and 9.7 s, respectively. With 120 cloudlets on 30 VMs, the conventional HGA and e-HGA required 16.06 s and 10.42 s, respectively, for task completion. Increasing the number of cloudlets to 150 on 30 VMs, the conventional HGA and e-HGA completed the task in 19.96 s and 10.15 s, respectively. For 180 cloudlets on 30 VMs, the conventional HGA and e-HGA achieved task completion in 21.86 s and 13.74 s, respectively.

Table 4 The 30-VM MakeSpan of the e-HGA against a conventional HGA

30 Virtual machines			
S/N		**MakeSpan**	
	Cloudlets	**HGA**	**e-HGA**
1	60	11.45	9.074
2	90	14.49	9.7
3	120	16.06	10.42
4	150	19.96	10.15
5	180	21.86	13.74
6	210	25.05	16.322
7	240	27.44	18.22
8	270	31.93	18.3
9	300	33.78	23.3
10	330	36.576	22.17

Similarly, with 210 cloudlets on 30 VMs, the conventional HGA and e-HGA required 25.05 s and 16.322s, respectively, for task completion. Furthermore, with 240 cloudlets on 30 VMs, the conventional HGA and e-HGA completed the task in 27.44 s and 18.22 s, respectively. With 270 cloudlets on 30 VMs, the conventional HGA and e-HGA achieved task completion in 31.93 s and 18.3 s, respectively. The completion time increased as 300 cloudlets were run on 30 VMs, with the conventional HGA and e-HGA requiring 33.78 s and 23.3 s, respectively, for task completion. Finally, with 330 cloudlets on 30 VMs, the conventional HGA and e-HGA completed the task in 36.576 s and 22.17 s, respectively. Consistent with previous findings, these results demonstrate that the proposed e-HGA generally outperforms the conventional HGA in terms of task execution time. Aziza and Krichen developed a GA based technique for modeling and solving a workflow-scheduling problem in cloud computing. The heuristic model and heterogeneous earliest finish time (HEFT) interfere in the formation of the starting population. Based on simulation findings utilizing real-world scientific process datasets, we show that the suggested technique outperforms existing HEFT and other strategies investigated in this research. In other words, investigations reveal that our suggested technique is highly efficient, making it potentially relevant for cloud WS. We created a GA-based module that was incorporated into the workflowSim framework, which is based on cloudSim [55]. *Table 4*, where the blue HGA bars are taller than the red e-HGA bars. The ascending order of the graph demonstrates that the experimental outcome increased together with the cloudlet number. The results also demonstrated that the proposed e-HGA performed better than the conventional HGA.

5. Discussion

Cloud WS differs significantly from scheduling in multiprocessor or grid systems due to its emphasis on utility. Unlike the latter two systems, where scheduling is primarily focused on meeting deadlines or minimizing workflow makeSpan, the "pay-as-you-go" cloud environment places equal importance on economic cost and performance. In the cloud, resources are commonly provisioned as VM, and virtualization technology serves as the basis for IaaS clouds. This technological approach sets cloud scheduling apart from utility grid computing [8]. Despite the widespread use of virtualization technology, it currently struggles to provide stable performance guarantees. Schad et al. [56] found a 24% overall CPU performance variability on Amazon's EC2 cloud, attributing it to resource sharing and competition among co-scheduled VMs on the same physical machine. This performance instability makes scheduling methods relying on task runtime estimations impractical in such environments. Consequently, new WS methods have been specifically developed for cloud environments. WS, a well-known NP hard problem, is extensively studied by researchers aiming to enhance workflow

execution performance. NP-hard problems are generally those that can be reduced to a different problem solved using polynomial time on a nondeterministic machine, such as optimization problems like the fractional knapsack or the traveling salesman.

In our experiments, we evaluated the performance of our e-HGA against a regular HGA using a variant of CloudLet. The experiments involved an increasing number of VM. The results demonstrated that the e-HGA outperformed the regular HGA, with the performance metric being makeSpan. MakeSpan is defined as the maximum time it takes for a single VM to complete all assigned tasks or jobs. For instance, in *Table 1*, where 15 cloudlet tasks were assigned to a VM, the makeSpan time for the conventional HGA was 9.668 seconds, while the e-HGA took 8.996 seconds. These results indicate the superior performance of the e-HGA compared to the normal HGA.

The graphical representation in *Figure 11*, with blue bars representing HGA and red bars representing e-HGA, further supports these findings. The ascending order of the graph illustrates that the experiment's outcome improves as the number of cloudlets increases. This study considered cloudlet characteristics with varying lengths, maintaining a constant network environment, and VM with different MIPS. Additionally, the e-HGA algorithm underwent testing with varying VMs and cloudlets (Tasks) to observe its behavior. Similar processes were repeated for the shortest job first and first-come-first-serve algorithm systems. The algorithms' performance was recorded based on the specified performance metrics. The study recommends further research to explore workflow schedulers using different types of HGA for optimizing workflow efficiency and resource management. WS is a well-known NP hard problem, which is being studied extensively by researchers to enhance the workflow execution performance. NP-hard generally refer to as problems that can be reduced to a different problem which are solved using a polynomial time on a nondeterministic machine such as optimization problem, Fractional Knapsack, Travel salesman etc. The proposed e-HGA collected tasks and mapped them evenly to the VMs. The results contributed to research on task scheduling optimization by scheduling task operations to reduce cost, enable efficient resource allocation, and manage time. Summary this study examined cloud computing and its concepts and subsequently considered cloudlet characteristics using different lengths with a constant network environment and VMs with different MIPS. The proposed e-HGA was tested with an increasing number of VMs to observe its behavior. The process was repeated using an increasing number of cloudlets. Based on the results, it was concluded that the number of cloudlets increased simultaneously with the increased number of VMs. Overall, the e-HGA performed better than the conventional HGA, and only seldom performed poorly in comparison to the conventional HGA. Therefore, it is recommended that further research be conducted in the area of makeSpan and cost of VM-scheduled task execution on cloud computer workflow schedulers using different HGAs to minimize cost and time in relation to the resources allocated for VM task execution. It is recommended that further research work should be carried out in the area of workflow scheduler using different types of HGA to further optimize the workflow result in the area of efficiency and resource management.

5.1Related research study

Based on the research conducted by Arif et al. [57] that introduced a machine learning-based downtime optimization (MLDO) strategy in 2016, which is an adaptive live migration technique based on predictive mechanisms that lowers downtime during live migration over wide area networks for normal workloads. Our key contribution is to use machine learning approaches to decrease downtime. Machine learning approaches are also employed in the prediction model, and adaptive threshold levels include automated learning. In terms of downtime noticed throughout the migration process, we compare our suggested strategy to existing solutions and find improvements of up to 15% [57]. A represented range of contemporary scientific difficulties using five procedures genetic algorithm-education and technology institute (GA-ETI) was tested and demonstrated its superiority against three well-known and up-to-date schedulers in this field (HEFT, provenance, and feature selective validation (FSV). Their investigation demonstrates that GA-ETI solutions have a shorter timeframe and lower monetary cost when compared to HEFT alternatives. Unlike FSV, GA-ETI generates a comprehensive scheduling configuration prior to execution that is of higher quality. Unlike Provenance, GA-ETI creates its own scheduling configuration and only uses a workflow manager system as a middleware to carry out scheduling choices [58]. GA-ETI also indicated that, contrary to popular belief, effective workflow execution does not need a large number of resources

(in comparison to the number of parallel nodes) in most circumstances. To continue this effort, the research intends to develop/incorporate cloud pricing models that take into account the variation of VM hiring costs during scheduling. Their research will also concentrate on performance oscillation in cloud systems and its influence on application execution [58]. As it was explained in the literature by [59] that identified limitations in the ant lion optimizer (ALO) and sine cosine algorithm (SCA) when applied to high complexity functions, as they tended to converge to local optima [59]. To address this issue, the researchers introduced a novel hybrid algorithm by combining ALO with SCA for multi-objective optimization in scheduling SWFs. The key innovation involved incorporating a greedy approach and introducing randomness based on chaos theory within a green cloud computing framework. The primary objectives of the algorithm were to minimize task makeSpan and cost, reduce energy consumption for a more environmentally friendly cloud computing environment, and enhance throughput. The researchers implemented their approach using the workflowSim simulator and compared the results with the strength Pareto evolutionary algorithm (SPEA) WS workflow algorithm. The outcomes demonstrated a notable reduction in both energy consumption and makeSpan, showcasing the efficacy of the proposed hybrid algorithm [59].

To produce an initial population, based on the research conducted by Aziza and Krichen (2020) that suggest a hybrid GA-based technique combined with HEFT [55]. They are searching for a solution that offers the optimal trade-off between time and cost while fulfilling the timeline and budget limits in their recommended strategy. Their model's primary function is to optimize the time required to conduct a group of interdependent operations in the cloud, lowering computational costs while meeting deadlines and budgets. To that end, they provide a hybrid strategy based on a GA for modeling and improving a workflow-scheduling problem in cloud computing. The HEFT interferes in the formation of the starting population [55]. Based on the results of their simulations utilizing real-world scientific process datasets, the suggested strategy outperforms existing HEFT and other strategies investigated in this study. In other words, experiments suggest that their proposed technique is highly efficient, making it potentially relevant for cloud WS. They created a GA-based module that was incorporated into the workflowSim framework, which is based on cloudSim [55]. Flower pollination algorithm (FPA) and GWO techniques proposed by [54] are employed as a hybrid employing PEFT algorithm for global and local optimization. The major purpose of the WS algorithm is to save time and money by utilizing VM migration [54]. In NP time, this method solves the subset problem and the choice problem. It works on the decision-making process to minimize server-side computation time and cost. This research suggests using hybrid optimization to improve the VM both locally and globally. The PEFT algorithm was used to initialize the system and functioned as a heuristic algorithm. This approach lowers the error associated with random optimization initialization. Flower pollination with GWO utilizing a hybrid technique yields significantly better end results than flower pollination with a GA. The suggested method additionally took into account the dependability parameter for various operations [54]. Introduce of an advanced HGA known as the e-HGA by [60], this novel approach combines the power of GAs with the efficiency of local search techniques. As a result, the e-HGA exhibits the capability to effectively navigate the solution space, preserve diversity, and converge towards high-quality scheduling solutions tailored for optimizing cloud workflows. A comparative analysis between the e-HGA and the conventional HGA demonstrated that the e-HGA outperformed the latter in terms of task completion speed across the majority of cases. In scenarios involving the execution of 50 and 70 cloudlets across ten VMs, it was observed that the conventional HGA outperformed the e-HGA in terms of execution speed. To illustrate, when 20 cloudlets were allocated to 10 VMs, the traditional HGA completed the task in 9.014 seconds, while the e-HGA achieved it in 8.698 seconds. As the count of both VMs and cloudlets increased concurrently, the traditional HGA consistently maintained its advantage over the e-HGA in terms of execution times. Our investigation leads to the conclusion that the performance of scheduling algorithms is notably influenced by the specific configuration of cloudlets and VMs.

5.2Performance analysis and discussion

In this segment, we conduct a comprehensive analysis of the performance of the proposed algorithm, e-HGA. The evaluation involves utilizing datasets with diverse characteristics, and the obtained results are compared against several selected algorithms, namely heuristics Microsoft certified professional (MCP) and HEFT, a generic evolutionary algorithm and recently introduced HGA and hybrid Self-Improved chimp optimization algorithm with glow swarm optimization algorithm

(HSCGS) The inclusion of these algorithms, which are based on different approaches, provides a solid foundation for studying and comparing the behavior of e-HGA. For the performance metric, we selected 30 VM. The VMs across all algorithms exhibit a 95% confidence interval for their corresponding values. This implies that for any workflow of a similar nature, the schedule length would fall within the given interval with 95% certainty. In some bar charts, the confidence interval may not be visually distinguishable from the mean value due to the scale used in those graphs (see *Figure 11*).

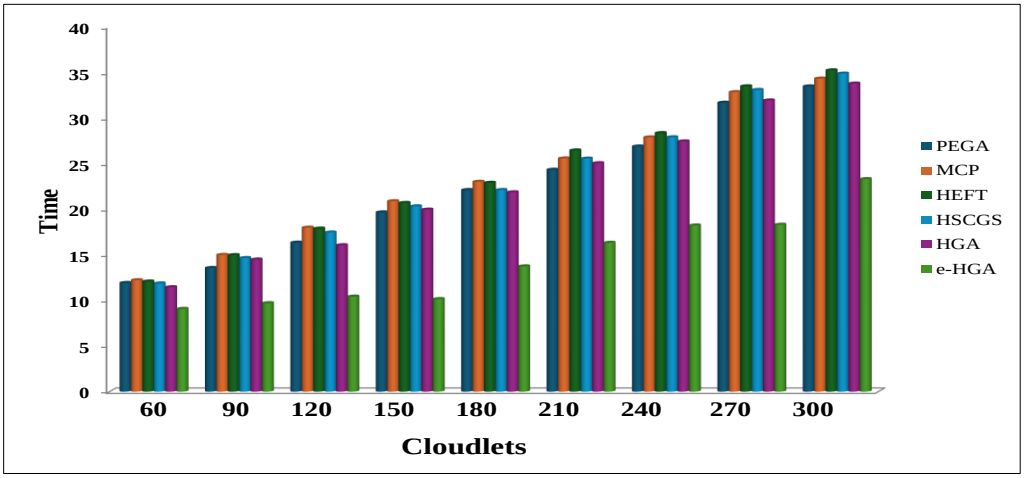

Figure 11 Performance of 30 Cloudlets at 100 population size

The proposed algorithm undergoes evaluation through simulations on a target system characterized by heterogeneity. Both resources and network links in the execution environment are heterogeneous. As tasks vary based on the workload type, the heterogeneity of execution nodes and tasks is taken into account in the heterogeneous execution times of each task on execution nodes. Similarly, the heterogeneity of network links and edges is implicitly considered through varying communication costs associated with the edges. Following numerous simulations, the most suitable parameters for the proposed algorithm are identified, yielding optimal results with crossover and mutation probabilities set at 0.8 and 0.02, respectively. To streamline the simulations, the population size and the number of generations are both set to 100.

A complete list of abbreviations is summarized in *Appendix I.*

6. Conclusion and recommendation

The proposed e-HGA collected tasks and mapped them evenly to the VMs. The results contributed to research on task scheduling optimization by scheduling task operations to reduce cost, enable efficient resource allocation, and manage time. Summary this study examined cloud computing and its concepts and subsequently considered cloudlet characteristics using different lengths with a constant network environment and VMs with different MIPS. The proposed e-HGA was tested with an increasing number of VMs to observe its behavior. The process was repeated using an increasing number of cloudlets. Based on the results, it was concluded that the number of cloudlets increased simultaneously with the increased number of VMs. Overall, the e-HGA performed better than the conventional HGA, and only seldom performed poorly in comparison to the conventional HGA. Therefore, it is recommended that further research be conducted in the area of makeSpan and cost of VM-scheduled task execution on cloud computer workflow schedulers using different HGAs to minimize cost and time in relation to the resources allocated for VM task execution. It is recommended that further research work should be carried out in the area of workflow scheduler using different types of GA to further optimize the workflow result in the area of efficiency and resource management.

Acknowledgment

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

Conflicts of interest

The authors have no conflicts of interest to declare.

Data availability

The data and code are owned by the Fundamental Research Grant Scheme (FRGS/1/2018/ICT03/UNISZA/02/01) under the Ministry of Higher Education (MOHE) and Universiti Sultan Zainal Abidin (UniSZA), Malaysia. They are confidential and cannot be released to third parties without the ministry's permission.

Author's contribution statement

Awolola Tejumola Busayo: Study conception and design, **Awolola Tejumola Busayo, Zarina Mohamad, Nor Aida Mahiddin, and Wan Nor Shuhadah Wan Nik**: Data collection, analysis and interpretation of results, **Awolola Tejumola Busayo**, **Zarina Mohamad, Nor Aida Mahiddin, and Wan Nor Shuhadah Wan Nik**: Draft manuscript preparation.

References

[1] Sun Q, Chien S, Hu D, Chen X. Optimizing customized transit service considering stochastic bus arrival time. Journal of Advanced Transportation. 2021; 2021:1-9.

[2] Tumuluru JS, Mcculloch R. Application of hybrid genetic algorithm routine in optimizing food and bioengineering processes. Foods. 2016; 5(4):1-13.

[3] Sulaiman M, Halim Z, Waqas M, Aydın D. A hybrid list-based task scheduling scheme for heterogeneous computing. The Journal of Supercomputing. 2021; 77:10252-88.

[4] Kaya SH, Corneille KV, Yassa S, Romain O, Etienne NM, Laurent BI. Industry 4.0 and industrial workflow scheduling: a survey. Journal of Industrial Information Integration. 2023: 100546.

[5] Kumari M, Singh V. Breast cancer prediction system. Procedia Computer Science. 2018; 132:371-6.

[6] Liu Y, Liu J, Zhu X, Wei D, Huang X, Song L. Learning task-specific representation for video anomaly detection with spatial-temporal attention. In international conference on acoustics, speech and signal processing 2022 (pp. 2190-4). IEEE.

[7] Karami S, Azizi S, Ahmadizar F. A bi-objective workflow scheduling in virtualized fog-cloud computing using NSGA-II with semi-greedy initialization. Applied Soft Computing. 2024; 151:111142.

[8] Abdel-basset M, Mohamed R, Abd EW, Sharawi M, Sallam KM. Task scheduling approach in cloud computing environment using hybrid differential evolution. Mathematics. 2022; 10(21):1-26.

[9] Zawawi O. Resource-efficient data pre-processing for deep learning (Doctoral Dissertation). Computer, Electrical and Mathematical Science and Engineering (CEMSE) Division. 2024.

[10] Bezdan T, Zivkovic M, Bacanin N, Strumberger I, Tuba E, Tuba M. Multi-objective task scheduling in cloud computing environment by hybridized bat algorithm. Journal of Intelligent & Fuzzy Systems. 2022; 42(1):411-23.

[11] Singh S, Kumar R, Singh D. An empirical investigation of task scheduling and VM consolidation schemes in cloud environment. Computer Science Review. 2023; 50:100583.

[12] Wu Z, Liu X, Ni Z, Yuan D, Yang Y. A market-oriented hierarchical scheduling strategy in cloud workflow systems. The Journal of Supercomputing. 2013; 63:256-93.

[13] Houssein EH, Gad AG, Wazery YM, Suganthan PN. Task scheduling in cloud computing based on meta-heuristics: review, taxonomy, open challenges, and future trends. Swarm and Evolutionary Computation. 2021; 62:100841.

[14] Zhao S, Miao J, Zhao J, Naghshbandi N. A comprehensive and systematic review of the banking systems based on pay-as-you-go payment fashion and cloud computing in the pandemic era. Information Systems and e-Business Management. 2023:1-29.

[15] Concha SL, Monzon BV. Harnessing the potential of emerging technologies to break down barriers in tactical communications. Telecom. 2023; 4(4):709-31.

[16] Huang J. The workflow task scheduling algorithm based on the GA model in the cloud computing environment. Journal of Software. 2014; 9(4):873-80.

[17] Abazari F, Analoui M, Takabi H, Fu S. MOWS: multi-objective workflow scheduling in cloud computing based on heuristic algorithm. Simulation Modelling Practice and Theory. 2019; 93:119-32.

[18] Zhu Z, Zhang G, Li M, Liu X. Evolutionary multi-objective workflow scheduling in cloud. IEEE Transactions on Parallel and Distributed Systems. 2015; 27(5):1344-57.

[19] Alzain MA, Pardede E, Soh B, Thom JA. Cloud computing security: from single to multi-clouds. In 45th Hawaii international conference on system sciences 2012 (pp. 5490-9). IEEE.

[20] Jensen M, Schwenk J, Bohli JM, Gruschka N, Iacono LL. Security prospects through cloud computing by adopting multiple clouds. In 4th international conference on cloud computing 2011 (pp. 565-72). IEEE.

[21] Krishna BH, Kiran S, Murali G, Reddy RP. Security issues in service model of cloud computing environment. Procedia Computer Science. 2016; 87:246-51.

[22] Yasrab R. Platform-as-a-service (PaaS): the next hype of cloud computing. arXiv preprint arXiv:1804.10811. 2018.

[23] Sadeeq MM, Abdulkareem NM, Zeebaree SR, Ahmed DM, Sami AS, Zebari RR. IoT and cloud computing issues, challenges and opportunities: a review. Qubahan Academic Journal. 2021; 1(2):1-7.

[24] Osanaiye O, Chen S, Yan Z, Lu R, Choo KK, Dlodlo M. From cloud to fog computing: a review and a conceptual live VM migration framework. IEEE Access. 2017; 5:8284-300.

[25] Wang L, Von LG, Kunze M, Tao J. Schedule distributed virtual machines in a service oriented environment. In 24th international conference on advanced information networking and applications 2010 (pp. 230-6). IEEE.

[26] Masdari M, ValiKardan S, Shahi Z, Azar SI. Towards workflow scheduling in cloud computing: a comprehensive analysis. Journal of Network and Computer Applications. 2016; 66:64-82.
[27] Żotkiewicz M, Guzek M, Kliazovich D, Bouvry P. Minimum dependencies energy-efficient scheduling in data centers. IEEE Transactions on Parallel and Distributed Systems. 2016; 27(12):3561-74.
[28] Rahman M, Hassan R, Ranjan R, Buyya R. Adaptive workflow scheduling for dynamic grid and cloud computing environment. Concurrency and Computation: Practice and Experience. 2013; 25(13):1816-42.
[29] Bala A, Chana I. A survey of various workflow scheduling algorithms in cloud environment. In 2nd national conference on information and communication technology 2011 (pp. 26-30).
[30] Rodriguez MA, Buyya R. A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. Concurrency and Computation: Practice and Experience. 2017; 29(8):e4041.
[31] Vincent FY, Redi AP, Hidayat YA, Wibowo OJ. A simulated annealing heuristic for the hybrid vehicle routing problem. Applied Soft Computing. 2017; 53:119-32.
[32] Saima GA. Workflow optimization in distributed computing environment for stream-based data processing model/Saima Gulzar Ahmad. Doctoral Dissertation, University of Malaya. 2017.
[33] Gul F, Mir I, Abualigah L, Sumari P. Multi-robot space exploration: an augmented arithmetic approach. IEEE Access. 2021; 9:107738-50.
[34] Daoud MI, Kharma N. A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks. Journal of Parallel and Distributed Computing. 2011; 71(11):1518-31.
[35] Srikanth M, Kessler JA. Nanotechnology-novel therapeutics for CNS disorders. Nature Reviews Neurology. 2012; 8(6):307-18.
[36] Seemakuthi S, Siriki VA, Lydia EL. A review on various scheduling algorithms. International Journal of Scientific & Engineering Research. 2015; 6:769-79.
[37] Zheng W, Sakellariou R. Budget-deadline constrained workflow planning for admission control. Journal of Grid Computing. 2013; 11(4):633-51.
[38] Zhao L, Ren Y, Sakurai K. Reliable workflow scheduling with less resource redundancy. Parallel Computing. 2013; 39(10):567-85.
[39] Zhong Z, Chen K, Zhai X, Zhou S. Virtual machine-based task scheduling algorithm in a cloud computing environment. Tsinghua Science and Technology. 2016; 21(6):660-7.
[40] Wei XJ, Bei W, Jun L. SAMPGA task scheduling algorithm in cloud computing. In 36th Chinese control conference 2017 (pp. 5633-7). IEEE.
[41] Lin R, Li Q. Task scheduling algorithm based on pre-allocation strategy in cloud computing. In international conference on cloud computing and big data analysis 2016 (pp. 227-32). IEEE.
[42] Fan Y, Liang Q, Chen Y, Yan X, Hu C, Yao H, et al. Executing time and cost-aware task scheduling in hybrid cloud using a modified DE algorithm. In computational intelligence and intelligent systems: 7th international symposium, Guangzhou, China, 2015 (pp. 74-83). Springer Singapore.
[43] Gupta N, Patel N, Tiwari BN, Khosravy M. Genetic algorithm based on enhanced selection and log-scaled mutation technique. In proceedings of the future technologies conference 2018 (pp. 730-48). Springer International Publishing.
[44] Wei H, Li S, Jiang H, Hu J, Hu J. Hybrid genetic simulated annealing algorithm for improved flow shop scheduling with makespan criterion. Applied Sciences. 2018; 8(12):1-20.
[45] Liaw CF. A hybrid genetic algorithm for the open shop scheduling problem. European Journal of Operational Research. 2000; 124(1):28-42.
[46] Oh IS, Lee JS, Moon BR. Hybrid genetic algorithms for feature selection. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2004; 26(11):1424-37.
[47] Lin CJ, Su SC. Protein 3D HP model folding simulation using a hybrid of genetic algorithm and particle swarm optimization. International Journal of Fuzzy Systems. 2011; 13(2):140-7.
[48] Calheiros RN, Ranjan R, De RCA, Buyya R. Cloudsim: a novel framework for modeling and simulation of cloud computing infrastructures and services. arXiv preprint arXiv:0903.2525. 2009.
[49] Buyya R, Murshed M. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and Computation: Practice and Experience. 2002; 14(13-15):1175-220.
[50] Wu F, Wu Q, Tan Y. Workflow scheduling in cloud: a survey. The Journal of Supercomputing. 2015; 71:3373-418.
[51] Sharifi M, Shahrivari S, Salimi H. PASTA: a power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources. Computing. 2013; 95(1):67-88.
[52] Hosseinzadeh M, Ghafour MY, Hama HK, Vo B, Khoshnevis A. Multi-objective task and workflow scheduling approaches in cloud computing: a comprehensive review. Journal of Grid Computing. 2020; 18:327-56.
[53] Radulescu A, Van GAJ. Fast and effective task scheduling in heterogeneous systems. In proceedings 9th heterogeneous computing workshop 2000 (pp. 229-38). IEEE.
[54] Khurana S, Singh R. Workflow scheduling and reliability improvement by hybrid intelligence optimization approach with task ranking. EAI Endorsed Transactions on Scalable Information Systems. 2019; 7(24):1-10.
[55] Aziza H, Krichen S. A hybrid genetic algorithm for scientific workflow scheduling in cloud environment. Neural Computing and Applications. 2020; 32:15263-78.

[56] Schad J, Dittrich J, Quiané-ruiz JA. Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proceedings of the VLDB Endowment. 2010; 3(1-2):460-71.
[57] Arif M, Kiani AK, Qadir J. Machine learning based optimized live virtual machine migration over WAN links. Telecommunication Systems. 2017; 64:245-57.
[58] Casas I, Taheri J, Ranjan R, Wang L, Zomaya AY. GA-ETI: an enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments. Journal of Computational Science. 2018; 26:318-31.
[59] Mohammadzadeh A, Masdari M, Gharehchopogh FS, Jafarian A. A hybrid multi-objective metaheuristic optimization algorithm for scientific workflow scheduling. Cluster Computing. 2021; 24:1479-503.
[60] Wood T, Ramakrishnan KK, Shenoy P, Van DMJ. CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines. ACM Sigplan Notices. 2011; 46(7):121-32.

Awolola Tejumola Busayo is a graduate of Al-Madinah International University, Kuala Lumpur, Malaysia, where he received his Bachelor of Science degree in Computer Science with a focus on Networking. His research interest was in preventing DDOS attacks through Identifier/Locator Separation. Currently, he is conducting research for his MSc on Optimizing Universal Workflow Schedulers using Hybrid Genetic Algorithms at Universiti Sultan Zainal Abidin, Kuala Terengganu, Malaysia.
Email: tejumolaawolola@yahoo.com

Zarina Mohamad (Phd) was born in Kuala Terenggnu, Terengganu, Malaysia in 1972. She received her B.S. and M.S. degrees in computer science in 2000 and 2004, respectively. In 2013 she received PhD from Universiti Tun Hussien Onn Malaysia. She is currently a senior lecturer in the Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin (UniSZA), Terengganu, Malaysia. Her research interests are Distributed Databases, Data Grids, Distributed Systems, Cloud Computing and Optimization.
Email: zarina@unisza.edu.my

Nor Aida Mahiddin (PhD) received a B.S. in Information Technology from National University of Malaysia, and M.S degree in computer science majoring in Distributed Computing and PhD degree in computer and information science from Auckland University of Technology, New Zealand. She is currently a senior lecturer in the Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin (UniSZA), Terengganu, Malaysia. Her research interests include Network Designs, Modelling and Performance Evaluation, Wireless Communication Networks, Disaster Resilient Network Design, Optimisation of Gateway Congestion Control, Ad Hoc and Sensor Networks and Wireless Mesh and Routing Protocols.
Email: aidamahiddin@unisza.edu.my

Wan Nor Shuhadah Wan Nik (PhD) is currently a senior lecturer in the Faculty of Informatics and Computing, University Sultan Zainal Abidin (UniSZA), Malaysia. She received a PhD in Computer Science (Distributed Systems) from University of Sydney, Australia in 2012 before appointed as a Deputy Director (Infrastructure & Services) at Information Technology Centre, UniSZA from year 2014 - 2017. She has been involved in more than ten research grants and led four national grants in the area of Distributed Systems. Her main research interest includes the area of Computer Networks and Distributed Systems, including Scheduling in Grid/Cloud and Utility Computing, Wireless Sensor Network, IoT, Heuristics and Optimization and Blockchain.
Email: wnshuhadah@unisza.edu.my

Appendix I

S. No.	Abbreviation	Description
1	ACO	Ant Colony Optimization
2	ALO	Ant Lion Optimizer
3	CPR	Constrained Pressure Residual
4	CPU	Central Processing Unit
5	e-HGA	Enhanced Hybrid Genetic Algorithm
6	EC2	Elastic Compute Cloud
7	FSV	Feature Selective Validation
8	FPA	Flower Pollination Algorithm
9	G&PSO	Greedy particle Swarm Optimization
10	GWO	Grey Wolf Optimization
11	GA	Genetic Algorithm
12	GA-ETI	Genetic Algorithm- Education and Technology Institute
13	HEFT	Heterogeneous Earliest Finish Time
14	HGA	Hybrid Genetic Algorithm
15	HSCGS	Self-Improved Chimp Optimization Algorithm with Glow Swarm Optimization Algorithm
16	IAAS	Infrastructure as a Service
17	LB-ACO	Load-Balancing Ant Colony Optimization
18	LIGO	Laser Interferometer Gravitational-Wave Observatory
19	MCP	Microsoft Certified Professional
20	MI	Million Instructions
21	MIPS	Million Instructions Per Seconds
22	MPGA	Multi-Population Genetic Algorithm
23	MLDO	Machine Learning-Based Downtime Optimization
24	NP	Nondeterministic Polynomial
25	NSGA-II	Non-dominated Sorting Genetic Algorithm II
26	PAAS	Platform as a Service
27	PACO	Pre-Allocation Ant Colony Optimization

28	PD	Protein Data
29	PEFT	The Parameter-Efficient Fine-Tuning
30	QOS	Quality of Service
31	SA	Simulated Annealing
32	SAAS	Software as a Service
33	SAMPGA	Self-Adaptive Multi-Population Genetic Algorithm
34	SCA	Sine Cosine Algorithm
35	SIPHT	Stanford Information Prediction Heterogeneous Tools
36	SLAs	Service Level Agreements
37	SPEA	Strength Pareto Evolutionary Algorithm
38	SWFs	Scientific Workflows
39	VMs	Virtual Machines
40	WMS	Workflow Management System
41	WS	Workflow Scheduling