A Comparative Study of Job Scheduling in Cloud Computing

Abdelbasset BARKAT[1]*, Zohair TAHRI[1], Derya YILTAS-KAPLAN[2]

[1] Laboratory of Informatics and its Applications of M'Sila (LIAM), Faculty of Mathematics and Computer Science, University of M'Sila, University Pole, Bordj Bou Arreridj Road, 28000 M'Sila, Algeria
abdelbasset.barkat@univ-msila.dz (**Corresponding author*), zohair.tahri@univ-msila.dz

[2] Istanbul University-Cerrahpaşa, Faculty of Engineering, Computer Engineering Department, B Block, Floor 6, Üniversite Mahallesi, 7 Üniversite Street, 34320 Avcilar, Istanbul, Türkiye
dyiltas@iuc.edu.tr

Abstract: In recent years, cloud computing has been widely adopted over the Internet due to the numerous advantages and services it offers to users. The resources in cloud computing are based on virtualization, which makes them dynamic and prone to frequent changes. These resources are utilized by cloud services or user applications in order to respond to user requests. The smallest unit of a user application, known as a job, requires exclusive access to certain resources which would enable its execution. Assigning jobs to cloud nodes (a set of computational resources) is a NP-complete problem, commonly referred to as job scheduling in the cloud. The aim of this paper is to propose an integer programming model for cloud job scheduling and to find an optimal or near-optimal solution by using two algorithms, namely a genetic algorithm called GAJSC, and a particle swarm optimization (PSO) algorithm. This study concludes with a comparison of the performance of these two approaches against certain traditional baseline algorithms, including the First-Come-First-Serve (FCFS) and Shortest Job First (SJF) algorithms in terms of the obtained makespan and their scalability.

Keywords: Cloud computing, Job scheduling, Optimization, Genetic algorithm, PSO, Cloudsim Plus, Virtualization, Makespan.

1. Introduction

Job scheduling in cloud computing refers to selecting the most suitable available resources for executing tasks or allocating computing machines in a way that minimizes the completion time as much as possible (Keivani & Tapamo, 2019). Cloud computing resources rely on virtualization, which makes them highly dynamic, and the set of available resources can change frequently. The cloud provides functionalities to clients as services and is characterized by three service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). On the other hand, users request services from the cloud to access these resources, where each service consists of a set of operations or jobs. Therefore, to ensure a good cloud performance, efficient job scheduling is essential.

Cloud computing is a model for enabling the ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with a minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models (Mell & Grance, 2011). On-demand self-service is a key characteristic of cloud computing, through which the users can independently provision computing resources - such as processing power, storage, or applications - whenever they need them, without requiring human interaction with the service provider. This gives clients the expectation of immediate responsiveness and availability whenever a request is made.

In addition to this, cloud service providers must be capable of serving multiple clients simultaneously in a manner that gives each user the impression of exclusive access to resources, similarly to using local services. This is enabled through resource pooling, where physical and virtual resources are dynamically allocated and reassigned according to demand. Furthermore, the cloud must accommodate varying workloads and scale rapidly in response to fluctuating user demands, a feature which is known as rapid elasticity. This means that the cloud service providers must respond efficiently to any number of requests, regardless of how many users are involved or how much computing power is required. These characteristics place great demands on the underlying infrastructure, making job scheduling a critical task in cloud computing. Effective job scheduling ensures that resources are allocated fairly, efficiently, and in a timely manner, directly influencing the cloud's ability to meet user expectations as responsiveness, scalability, and seamless performance are concerned.

https://doi.org/10.24846/v34i3y202508

The job scheduling algorithms are classified into two classes (Santhosh et al., 2016). The first class includes batch mode heuristic scheduling (BMHA) algorithms, which wait a fixed period of time for job to arrive. Each newly arrived job is pushed into a queue to maintain the order of arrival, then the scheduling process is started. The second class includes online mode heuristic scheduling (OMHS) algorithms, here the jobs are scheduled when they arrive, and there is no awaiting time, this mode is more appropriate for the cloud environment.

The objective of this research is to propose two novel approaches for job scheduling in cloud environments: one approach is based on a genetic algorithm, and the other employs the well-known PSO optimization technique, both aiming to achieve results that surpass the current state of the art.

The remainder of this paper is organized as follows. The related works about job scheduling in cloud computing are presented in Section 2, while Section 3 is dedicated to formulating the analysed problem in a suitable way, by means of an integer programming-based model. Further on, Section 4 sets forth the proposed job scheduling solutions (one based on a genetic algorithm and the other on the PSO technique) and Section 5 discusses the experiments which were carried out and the obtained results. Finally, Section 6 concludes this paper and outlines possible future research directions.

2. Related Work

The goal of a scheduler in a cloud computing environment is to utilize the resources efficiently and minimize the job execution time (Wei et al., 2018). A job scheduling problem is considered NP-hard if there is no exact algorithm that can guarantee an optimal solution in a reasonable amount of time. To address this challenge and achieve the desired objectives, many researchers have proposed a variety of methods and algorithms, most of which are heuristic-based ones, as heuristics is well-suited for solving complex problems where an exhaustive search is impractical. In this sense, the works of Sanjalawe et al. (2025), Houssein et al. (2021) and Murad et al. (2022) can enable a more detailed understanding and in-depth discussion of this subject.

The work of Lipsa et al. (2023) introduces an innovative priority-based job scheduling approach for an efficient task scheduling in cloud computing environments. This method combines several advanced techniques for optimizing both the task prioritization and resource utilization, addressing key challenges in cloud workload management. Its core innovation lies in a novel priority assignment mechanism using a specialized matrix that considers both the task size and the estimated execution time. This intelligent prioritization system works in tandem with a Fibonacci heap data structure, enabling rapid task insertion and retrieval operations while maintaining an optimal priority ordering. The scheduling architecture employs a hybrid parallel processing model that strategically blends non-preemptive and preemptive strategies. To prevent task starvation, the system implements a dynamic priority escalation mechanism that gradually increases the priority of pending tasks. The theoretical foundation uses the M/M/n queuing model to minimize the waiting times, processing times, and transmission delays. The experimental results demonstrate significant improvements over the existing solutions like BATS and IDEA, particularly in handling large-scale workloads (up to 10,000 tasks). For more details about priority-based job scheduling in the cloud, the reader is invited to consult the works of Murad et al. (2022, 2024).

The study of Sutar et al. (2024) introduces an innovative approach to cloud job scheduling that simultaneously addresses energy efficiency and cost reduction - two often competing priorities in data centre management. The authors developed a dual-objective optimization model using NSGA-II, an advanced evolutionary algorithm, to intelligently allocate jobs across virtual machines (VMs) while minimizing both power consumption and operational expenses. This framework incorporates dynamic energy measurement through the Dynamic Voltage and Frequency Scaling (DVFS) technology and comprehensive cost modeling, providing a more realistic solution than the traditional single-objective methods. Rigorous testing using CloudSim simulations demonstrates significant improvements, as this

model achieved energy savings of up to 40% and a cost reduction of 30% in comparison with the existing algorithms like the PSO and ABC optimization algorithms. And, although the results are promising, the authors acknowledge there are limitations related to the current simulations and propose relevant future directions, including the real-time adaptation of job scheduling. This work represents a meaningful step towards sustainable cloud computing, proving that the environmental and economic objectives can be successfully balanced through an intelligent system design.

Sridhar & Babu (2015) claim that the PSO algorithm works well for a global search but not so well for a local search, unlike the Tabu Search (TS) algorithm, which outperforms PSO in the context of the local search, therefore they proposed a Hybrid Particle Swarm Optimization algorithm for job Scheduling in Cloud Computing, taking into account how the weakness of the PSO algorithm in the local search is complemented by the Tabu search in order to increase the probability of finding an optimal solution. The algorithm starts by initializing a random population of particles, then the fitness of each particle is calculated using a fitness function. Further on, the algorithm divides the population randomly into two halves, one of which is explored by the PSO algorithm, and the other one by the Tabu Search algorithm, to finally combine the two halves, and update the "pbest", the "gbest" particles and the Tabu list. These steps were repeated until the termination condition was verified.

The objective of the work of Zhu et al. (2021) is to develop and evaluate an efficient task scheduling method for multi-cloud environments that optimizes both the makespan and total cost while satisfying the security and reliability constraints. The proposed scheduling algorithm is called Matching and Multi-round Allocation (MMA) and it includes three steps:

1. Matching phase: For each task, the algorithm finds virtual machines (VMs) that meet its security, reliability, and performance needs. It picks the VM that best matches each task;

2. Initial allocation: Tasks are sorted by priority and size, and then assigned to their best matching VMs. This helps balance the load across similar VMs;

3. Multi-round allocation: Tasks are reallocated in several rounds to reduce the differences in completion time among VMs, which improves the efficiency of the process and reduces the total execution time.

A Cloud Manager monitors the availability of virtual machines (VMs) and coordinates the scheduling process. As a centralized management centre, it has access to multiple cloud environments.

Researchers have tried many ways to improve cloud scheduling - some focus on task order, others on combining different algorithms, and a few on trying to save energy and costs. While these methods work well in specific cases, none of them can handle all situations perfectly. This paper aims to create a more flexible scheduler that works efficiently in different cloud environments.

3. Problem Definition

As shown in Figure 1, the cloud-based job scheduling framework has three main parts: the User Portal, Job Scheduler and Management Module. The job scheduler is the brain of the system, it decides where and when to run each job by assigning it to the right VM (Santhosh et al., 2016).

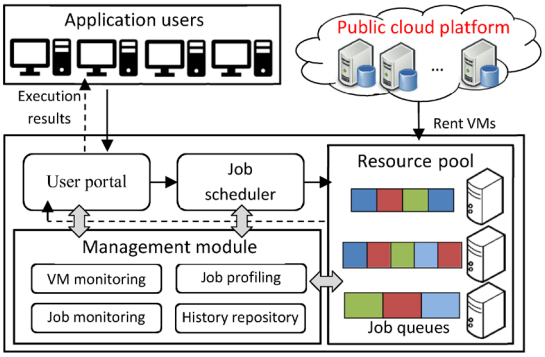

Figure 1. Cloud-based job scheduling framework (Santhosh et al., 2016)

In general, the users connect to the cloud to execute their requests by exploiting a set of cloud resources, and the requests are considered as a set of jobs (or tasks) $T = \{t_0, t_1. . . .t_{n-1}\}$, such that each job is a pair $t_i = \{l_i, p_i\}$, where l_i represents the job duration measured by the number of cycles, and p_i is the priority of the job. The cloud resources are denoted by $CR = \{CR_0, CR_1. . . CR_{m-1}\}$, and each cloud resource is characterized by a computing capacity

measured by the number of cycles per unit of time and a cost representing the total resource usage cost incurred when tasks are assigned to a cloud resource in a cloud computing environment $CR_i = \{S_i, C_i\}$. It means that each job t_i has a processing time in a cloud resource CR_j equal to $PT_{ij} = l_i/S_j$. In order to simplify the model, the setup and transfer times are ignored. The job scheduling consists in mapping $T = \{t_0, t_2. . .t_{n-1}\}$ to $CR = \{CR_0, CR_1. . . CR_{m-1}\}$ so as to minimize the job completion time.

The objective of the problem of job scheduling in the cloud environment is to minimize the makespan which can be defined as the maximum cost of any machine (Feldman et al., 2025).

Task assignment to cloud resources can be performed in various ways, and the effectiveness of each method can be evaluated by calculating the makespan (the total time required to complete all tasks). For example, a set of cloud resources $CR = \{CR_0(2, 3), CR_1(4, 2), CR_2(3, 1)\}$ and a set of tasks $T = \{t_0(9, 2), t_1(8, 3), t_2(2, 1), t_3(6, 5), t_4(8, 3), t_5(2, 2)\}$ are given. Table 1 illustrates two different job assignment strategies: the former is based on a random assignment with a makespan of 8.5, while the latter is based on an optimal assignment with a reduced makespan of 4.

Table 1. Example of job scheduling in the cloud

Random allocation makespan = 8.5			
$CR_0(2, 3)$	$t_0(9, 2) => PT_{00} = 4.5$	$t_1(8, 3) => PT_{10} = 4$	8.5
$CR_1(4, 2)$	$t_2(2, 1) => PT_{21} = 0.5$	$t_3(6, 5) => PT_{31} = 1.5$	2
$CR_2(3, 1)$	$t_4(8, 3) => PT_{42} = 2.33$	$t_5(2, 2) => PT_{52} = 0.66$	2.99

Optimized allocation makespan = 4			
$CR_0(2, 3)$	$t_2(2, 1) => PT_{20} = 1$	$t_3(6, 5) => PT_{30} = 3$	4
$CR_1(4, 2)$	$t_1(8, 3) => PT_{11} = 2$	$t_4(8, 3) => PT_{41} = 2$	4
$CR_2(3, 1)$	$t_0(9, 2) => PT_{02} = 3$	$t_5(2, 2) => PT_{52} = 0.66$	3.66

Integer Programming (IP) is a mathematical optimization technique employed for solving problems in which some or all decision variables must take integer values. It is particularly well-suited for modeling decision-making problems involving discrete choices, such as job assignment, job scheduling, or resource allocation. Further on, an IP formulation for the analysed job scheduling problem is presented, using binary variables to represent the assignment of jobs to virtual machines.

The core component of this model is the assignment matrix A, where rows represent the available virtual machines (CRs) and columns represent jobs. Each element at the intersection of row (i) and column (j) indicates whether job j is assigned to CR_i, and is defined as:

$$a_{ij} = \begin{cases} 1 \text{ if } t_i \text{ is assigned to CR } j \\ 0 \text{ otherwise} \end{cases} \quad (1)$$

According to the IP specification, a_{ij} must take an integer value. In addition, and in order to ensure that each job is assigned to one and only one cloud resource, the following statement must be verified:

$$\sum_{j=0}^{|CR|-1} a_{ij} = 1 \quad \forall \; i = 0...|T| \quad (2)$$

The function which represents the value of the makespan is given by equation (3):

$$MS = \max_{j=0...|CR|-1} \left(\sum_{i=0}^{|T|-1} a_{ij} * P_{ij} \right) \quad (3)$$

Further on, the total cost of the current job scheduling is given by equation (4):

$$\text{Cost} = \sum_{j=0}^{|CR|-1} \sum_{i=0}^{|T|-1} a_{ij} * P_{ij} * C_j \quad (4)$$

The objective function will be a weighted sum function, the aim is to minimize the sum of the makespan and the total cost, each multiplied by a corresponding weight factor that reflects its relative importance in the optimization process as expressed in equation (5):

$$Minimize\left(w_1 * MS + w_2 * \text{Cost}\right) \quad (5)$$

where w_1 represents the importance of minimizing the job execution time. and w_2 the importance of minimizing the total cost.

If one chooses to ignore the cost in the scheduling process, the objective function will only minimize the makespan, and vice versa - if the makespan is ignored, it will only minimize the cost. In this paper, the focus is on minimizing the makespan, which is the total time required to complete all the assigned jobs. This objective is critical for improving the resource utilization and reducing the overall execution time in cloud environments. Other important objectives, such as minimizing the total cost or considering a multi-objective

approach that balances both the overall execution time and cost, are beyond the scope of this work and shall be left for future research.

4. The Proposed Method

Based on the previously described model, this section presents two solutions as methods for the job scheduling problem. One method is based on a genetic algorithm, and it is referred to as the Genetic Algorithm of Job scheduling in the Cloud (GAJSC), and the other is an adaptation of PSO algorithm tailored for addressing the same scheduling problem. Both approaches use the problem representation described earlier to find efficient job-to-VM assignments.

4.1. Genetic Algorithm of Job Scheduling in the Cloud (GAJSC)

This section presents the first proposed algorithm, which is a genetic algorithm for job scheduling, based on the problem definition provided in the previous section. The complete steps of Algorithm 1 are illustrated below:

Algorithm 1. Genetic algorithm for job scheduling

Input: crList, jobList, POP SIZE, MAX GENERATIONS, CROSSOVER RATE, MUTATION RATE
Output: best-Solution
• Initialize population with random assignments.
• Evaluate initial population and set random best-Solution.
for $i \leftarrow 1$ **to** *MAX GENERATIONS* **do**
  for $j \leftarrow 1$ **to** *POP SIZE* **do**
  • Calculate fitness for *individual[j]*
  • Update best-Solution if *individual[j]* better than old best-Solution.
  end
• Select parents using tournament selection.
• Perform crossover and mutation to create new population.
• Replace the current population with the new population.
end

In addition to the standard inputs of the algorithm, such as the population size and the maximum number of generations, the algorithm also takes three problem-specific inputs: the set of cloud resources, the set of jobs, and an initial (random) assignment.

By encoding the solutions as mentioned above, the problem is modeled by using an assignment matrix where rows represent virtual machines (VMs) and columns represent jobs. A value of 1 at the intersection of a row and a column means that a job is assigned to a VM. In the framework of the genetic algorithm, and in order to encode the solution, this matrix is converted into a one-dimensional array. The size of the array corresponds to the number of jobs. Each element in the array represents a job, where its index is the job ID and the value of an element is the ID of the VM assigned to it (see Figure 2). This encoding was chosen because it makes genetic algorithm operations easier. For example, in the crossover step, a random point was picked and parts of two parent solutions were swapped to create two children - each child getting part of its data from each parent. In the mutation step, a position was randomly selected in the array and its value was changed within the allowed range.

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
VM_1	0	1	0	0	0	0	0	1	0	0
VM_2	0	0	0	1	0	0	0	0	0	1
VM_3	1	0	1	0	0	0	0	0	0	0
VM_4	0	0	0	0	1	0	1	0	0	0
VM_5	0	0	0	0	0	1	0	0	1	0

Assignment matrix

J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
3	1	3	2	4	5	4	1	5	2

Assignment array

Figure 2. Solution encoding for GAJSC

Table 2 includes the key parameters of the proposed GAJSC, which were used in the experimental phase. Among these parameters, the maximum number of generations is fixed for experiments with the real dataset, while for the synthetic dataset it is variable for evaluating the algorithm under varying conditions.

Table 2. GAJSC's parameters

Parameter	Value
Population size	50
Maximum number of generations	500
Selection method	Tournament selection
Crossover probability	0.8
Mutation probability	0.02
Crossover operator	Four points
Mutation operator	Randomly change
Encoding scheme	Integer

4.2. Particle Swarm Optimization for Job Scheduling

For the PSO-based method, the same solution encoding is used as for the previous algorithm because it is easier to implement and maintain during the programming phase. It is also important to note that the proposed PSO algorithm (Algorithm 2), is a discrete version, adapted to match the discrete nature of the job scheduling problem.

Algorithm 2. PSO for job scheduling

Input: crList, jobList, swarm size, max iter, w, c_1, c_2
Output: gBest: Optimal job-to-resource assignment
for $i \leftarrow 0$ **to** *swarm size* **do**
  for $j \leftarrow 0$ **to** *jobList.lenght* **do**
    • position[i][j] = crList[random(0,crList.length)]
    • velocity[i][j] = random (0, 1)
    • makespan = compute makespan(crList, jobList, position)
    • fitness[i][j] = 1 / (makespan + 1)
  end
  if *pBest[i] is worse than position[i]* **then**
    • pBest[i] = position[i]
    • pBest fitness[i] = fitness[i]
  end
end
/* Initialize global best */
for $i \leftarrow 0$ **to** *swarm size* **do**
  if *gBest is worse than position[i]* **then**
  • gBest = position[i]
  • gBest fitness = pBest fitness[i]
  end
end
/* PSO Main Loop */
for $i \leftarrow 0$ **to** *max iter* **do**
  for $j \leftarrow 0$ **to** *swarm size* **do**
    for $k \leftarrow 0$ **to** *jobList.lenght* **do**
    • $r_1, r_2 = random(0, 1)$
    • $cognitive = c_1 * r_1 * (pBest[j][k] != position[j][k])$
    • $social = c_2 * r_2 * (gBest[j] != position[j][k])$
    • $velocity[j][k] = w * velocity[j][k] + cognitive + social$
    • $velocity[j][k] = clamp(velocity[j][k], 0, 1)$
    if $random(0, 1) < 0.5$ **then**
    • position[j][k] = pBest[j][k]
    else
    • position[j] = gBest;
    end
    /* Evaluate new position */
    • new makespan = compute makespan(jobs, resources, position)
    • new fitness = 1 / (new makespan + 1)
    /* Update personal best
    if *new fitness > pBest fitness*[*i*] **then**
    • pBest[j] = position[j]
    • pBest fitness[j] = new fitness
    end
    /* Update global best */
    if *new fitness > gBest fitness* **then**
    • gBest = position[j]
    • gBest fitness = new fitness
    end
  end
end
return *gBest*

Table 3 presents the key parameters of the proposed PSO. The maximum number of iterations is fixed at 500 for experiments with the real dataset, whereas for the synthetic dataset this parameter is adjusted to assess the algorithm's performance under varying conditions.

Table 3. PSO's parameters

Parameter	Value
Population size	50
Max. number of iterations	500
Inertia weight (w)	0.7
Cognitive coefficient (c_1)	1.4
Social coefficient (c_2)	1.4
Velocity limits (V_{max})	Random

5. Experiments and Analysis

The simulation was conducted on a machine running the Windows operating system and powered by a 10th-generation Intel i5 processor, and all the algorithms in this study were implemented using Java as a programming language based on the CloudSim plus framework (Silva Filho et al., 2017). With regard to experimental data, two types of datasets were used, namely a synthetic dataset and a real-world dataset.

5.1. Synthetic Dataset

In order to test the set of algorithms presented in this paper, a synthetic dataset was initially used for evaluating their performance before applying them to a real-world dataset. The synthetic dataset was generated for simulations using the CloudSim Plus framework. It consists of a set of configurations (problems P_1, P_2 and P_3), each of them including two files: one for virtual machines (VMs) and one for jobs (cloudlets). Table 4 includes a summary of the three problems used in this paper.

Table 4. The three problems used in this paper

Problem	files	
P_1	VMs = 5	Cloudlets = 100
P_2	VMs = 10	Cloudlets = 500
P_3	VMs = 200	Cloudlets = 10000

For the VM configuration file, each virtual machine is defined by two parameters: a sequential

ID and its processing power, measured in tab (MIPS). The cloudlet configuration file contains job specifications, where each job is characterized by its length and a priority value.

Figures 3 to 5 illustrate the convergence behavior of both PSO and GA for the cloud job scheduling problem (Figure 3 for problem P_1, Figure 4 for P_2, and Figure 5 for P_3). As expected, both algorithms exhibit a decreasing makespan with an increasing number of iterations, indicating that an increasing number of iterations allows the algorithms to converge toward better solutions.

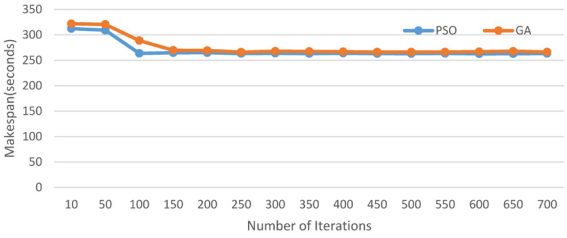

Figure 3. The convergence behavior of both PSO

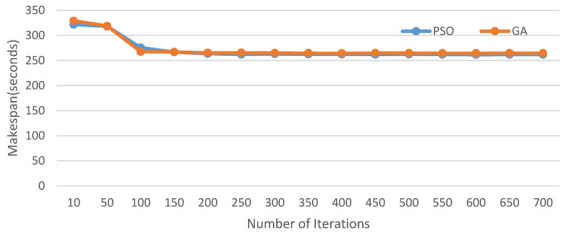

Figure 4. The convergence behavior of both PSO and GA for the problem P_2

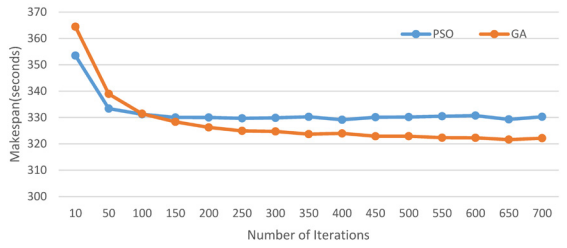

Figure 5. The convergence behavior of both PSO and GA for the problem P_3

This behavior is consistently observed across all the three versions of the analysed problem. However, for Problem 3 (P_3), which represents the largest and most complex scenario, the GA demonstrates a slight advantage over PSO in terms of minimizing the makespan. Table 5 represents the best solution found by the GA for the problem P_1.

Table 5. The solution of the problem P_1 found by GAJSC

Virtual machine	Assigned cloudlets
VM_1	3, 6, 12, 13, 15, 18, 23, 29, 37, 38, 40, 44, 45, 46, 49, 51, 55, 61, 65, 72, 73, 76
VM_2	0, 1, 8, 10, 11, 17, 34, 36, 48, 53, 56, 58, 63, 85, 89, 91, 95, 96, 98
VM_3	2, 4, 20, 22, 26, 31, 33, 41, 43, 57, 64, 70, 77, 80, 90, 93, 94
VM_4	14, 16, 25, 27, 28, 32, 35, 47, 50, 59, 60, 69, 74, 78, 82, 86, 92, 99
VM_5	5, 7, 9, 19, 21, 24, 30, 39, 42, 52, 54, 62, 66, 67, 68, 71, 75, 79, 81, 83, 84, 87, 88, 97

The plot in Figure 6 compares the execution time for PSO and the Genetic Algorithm (GAJSC) across various numbers of iterations. It clearly shows that both algorithms experience an increase in the execution time as the number of iterations grows, which is expected. However, PSO demonstrates a significantly steeper increase in the execution time in comparison with GA. At every iteration level, the GA consistently requires less time than PSO.

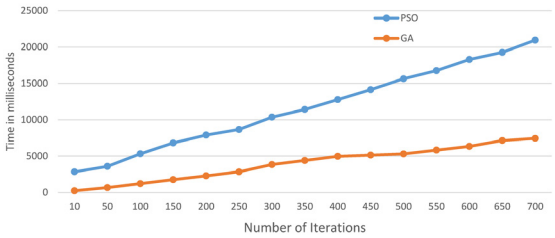

Figure 6. Comparison of the execution time for PSO and the Genetic Algorithm (GAJSC)

5.2. Real-world Dataset

The Google Cloud Jobs (GoCJ) Dataset is a public dataset made available to help researchers and practitioners understand and simulate large-scale job scheduling workloads in cloud computing environments. It is designed to reflect the behavior of real jobs running on Google's data centers, offering insights into production-level job scheduling, resource usage, and task execution (Hussain & Aleem, 2018).

The line chart in Figure 7 illustrates the makespan performance (in seconds) of four scheduling algorithms - PSO, the FCFS (First-Come-First-Serve) algorithm, the SJF (Shortest Job First) algorithm, and the genetic algorithm (GAJSC) which were evaluated across different configurations of the GoCJ (Google Cloud Jobs) dataset. Each dataset configuration (e.g. "100-5") represents a unique combination of jobs and virtual machines, plotted along the

horizontal axis, while the vertical axis indicates the corresponding makespan.

At a glance, both the GA and PSO demonstrate a superior performance, achieving significantly lower makespan values in comparison with the SJF and FCFS algorithms. Their trend lines appear very close throughout the entire range of datasets, which may obscure a key insight: GA consistently performs slightly better than PSO, a finding that aligns with the previous simulations. Despite their close values, GA's makespan is marginally lower than PSO's makespan across nearly all datasets, underscoring its higher optimization capability, which is particularly important for complex or evolving cloud workloads.

In contrast, FCFS shows the poorest performance, with the highest makespan values throughout all datastes. This result highlights FCFS's inefficiency in dynamic scheduling environments, where it fails to adapt to workload complexity. SJF performs moderately, offering better outcomes than FCFS but falling short of the optimization achieved by GA and PSO. Table 6 presents the detailed results corresponding to the visual summary shown in Figure 7.

Table 6. Makespan-based comparison of the scheduling algorithms

Dataset	FCFS	SJF	PSO	GAJSC
100-5	3344.00	3033.33	2512.78	2507.08
150-7	5280.63	4506.25	3381.88	3383.75
200-10	4885.63	3667.50	2360.36	2352.50
250-12	3711.25	3135.63	2325.56	2319.17
300-15	4589.50	3257.22	2605.00	2481.67
350-17	4202.00	3879.38	2847.92	2682.27
400-20	4668.33	3390.56	2547.14	2460.50
450-22	3776.67	3162.78	2398.64	2359.09
500-25	5862.50	3370.00	2752.78	2687.31
550-27	4165.00	3500.63	2774.00	2692.86
600-30	4075.00	3724.38	3045.00	2862.92
650-32	4863.13	3416.67	3036.92	2868.57
700-35	4118.13	3772.50	2821.25	2675.00
750-37	3994.50	3796.25	3042.92	2953.00
800-40	3930.00	3725.00	2630.00	2571.25
850-42	4025.00	3382.22	2683.50	2623.64
900-45	5212.22	3816.25	3170.50	3003.93
950-47	4939.38	3576.88	3059.44	2757.00
1000-50	5262.50	3793.75	2981.50	2870.00

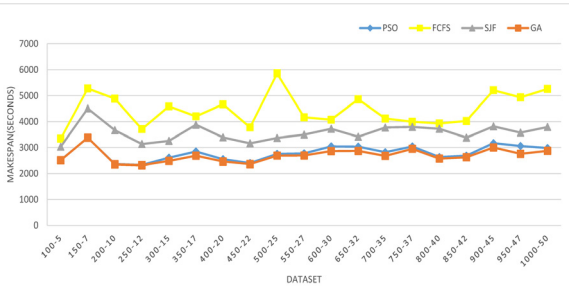

Figure 7. Comparison of the four employed Scheduling Algorithms applied to the Google cloud dataset

Table 7 summarizes the characteristics of the four proposed scheduling algorithms, namely the First-Come-First-Served (FCFS) and the Shortest Job First (SJF) algorithms in comparison with the proposed Particle Swarm Optimization (PSO) and Genetic Algorithm for Job Scheduling in Cloud (GAJSC). The comparison is based on key criteria such as the average makespan, consistency, scalability, and the most appropriate use cases for each approach. This qualitative assessment highlights the strengths and limitations of each algorithm, offering insights into their applicability under different workload and system conditions.

In order to benchmark the proposed methods against the related approaches, the Average Resource Utilization Ratio (ARUR) was used as the primary metric for comparison. The ARUR is defined as the average resource utilization by each method during the complete execution of all the cloudlets on the available VMs as given in equation (6) (Ibrahim et al., 2020):

$$ARUR = \frac{avg(makespan)}{\max(makespan)} \tag{6}$$

Table 8 presents the efficiency of the proposed approaches in comparison with two other related methods in terms of ARUR. All the approaches considered in this comparison were evaluated using the GoCJ dataset for job scheduling, along with synthetic data.

Table 7. Comparison of the Scheduling Algorithms Across Multiple Metrics

Algorithm	Average Makespan	Consistency	Scalability	Best Use Cases
PSO	Lowest	High	Excellent	Real-time cases
GAJSC	Very Low	High	Excellent	Complex, evolving workloads
FCFS	Highest	Low	Low	Simplicity over efficiency
SJF	Moderate	Moderate	Moderate	Small job workloads

Table 8. Comparison of the Scheduling Algorithms in term of ARUR

Method	Synthetic data	GoCJ dataset
MOABCQ_FCFS	0.742	0.796
MOABCQ_LJF	0.801	0.792
PSO	0.900	0.824
GAJSC	0.896	0.794

The method MOABCQ is a multi-objectives method for solving the problem of job scheduling in cloud computing based on the artificial bee colony algorithm adopted along with the implementation of the Q-learning algorithm. To extend this approach, the First-Come-First-Serve heuristic is incorporated to create MOABCQ_FCFS, while the Largest-Job-First heuristic is applied to derive the MOABCQ_LJF algorithm (Kruekaew & Kimpan, 2022).

The results show that the proposed methods significantly outperform the related approaches in terms of ARUR. While GAJSC exhibits strong results for synthetic data, its performance on the GoCJ dataset is comparable with that of the existing methods. In contrast, PSO consistently achieves the highest ARUR across both datasets, demonstrating its effectiveness and robustness in resource utilization. Although GAJSC a consistently achieves a better makespan than PSO algorithm, the ARUR results indicate that PSO utilizes resources more efficiently. This suggests that GA prioritizes the reduction of the job completion time, possibly at the expense of leaving some resources idle, whereas PSO distributes the workload more evenly, leading to a higher resource utilization but slightly longer job completion times.

6. Conclusion

Cloud computing makes it possible to access shared resources like servers, storage, and applications over the Internet whenever needed. A key part of this is virtualization, which lets one physical server run multiple virtual machines (VMs), thereby using hardware more efficiently. Thanks to this, organizations can set up and manage their IT systems more flexibly through models such as IaaS, PaaS, and SaaS. The main advantages include lower upfront costs, the ability to quickly adjust resources as needed, and better options for disaster recovery. Leading companies like AWS, Microsoft Azure, and Google Cloud use virtualization in order to deliver these services on a global scale.

In highly dynamic cloud environments where the amount of resources changes every second, there is a critical need for efficient job scheduling strategies. This paper addresses the task scheduling problem in such environments by proposing two novel algorithms. One of them is a genetic algorithm-based approach, referred to as GAJSC (Genetic Algorithm for Job Scheduling in Cloud), and the other is a meta-heuristic algorithm based on Particle Swarm Optimization (PSO). Both algorithms were evaluated using synthetic and real-world datasets and compared against two traditional baseline algorithms, namely FCFS and SJF. The experimental results demonstrate that the PSO algorithm has a higher resource utilization, resulting in higher ARUR values.

However, it does not consistently achieve the lowest makespan, while GAJSC features a balanced performance, combining a competitive makespan with a high ARUR which makes it a particularly effective solution for scheduling in highly dynamic cloud infrastructures.

As a continuation of this research, future work could focus on extending the current job scheduling framework for supporting a multi-objective optimization. In real-world cloud environments, job scheduling decisions often involve trade-offs between multiple conflicting objectives such as the execution time, resource utilization, energy consumption, and cost efficiency. The aim of future work will be to integrate dynamic priority adjustment mechanisms for adapting to real-time workload changes and service-level agreements (SLAs). This extension will enable a more comprehensive and practical scheduling strategy that better aligns with the complex and heterogeneous nature of modern cloud computing infrastructures.

REFERENCES

Feldman, M., Garg, J., Narayan, V. V. et al. (2025) Proportionally Fair Makespan Approximation. In: The *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI-25)*, *27 February - 4 March 2025, Philadelphia, USA*. Washington, USA. IEEE. 13839-13846.

Houssein, E. H., Gad, A. G., Wazery, Y. M. et al. (2021) Task Scheduling in Cloud Computing based on Meta-heuristics: Review, Taxonomy, Open Challenges, and Future Trends. *Swarm and Evolutionary Computation*. 62, art. ID 100841. https://doi.org/10.1016/j.swevo.2021.100841.

Hussain, A. & Aleem, M. (2018) GoCJ: Google Cloud Jobs Dataset for Distributed and Cloud Computing Infrastructures. *Data*. *3*(4), art. no. 38. https://doi.org/10.3390/data3040038

Ibrahim, M., Nabi, S., Baz, A. et al. (2020) Towards a Task and Resource Aware Task Scheduling in Cloud Computing: An Experimental Comparative Evaluation: *International Journal of Networked and Distributed Computing*. 8(3), 131. https://doi.org/10.2991/ijndc.k.200515.003.

Keivani, A. & Tapamo, J.-R. (2019) Task Scheduling in Cloud Computing: A Review. In: *2019 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, *5-6 August 2019, Winterton, South Africa*. New York, USA, IEEE. https://doi.org/10.1109/ICABCD.2019.8851045.

Kruekaew, B. & Kimpan, W. (2022) Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm With Reinforcement Learning. *IEEE Access*. 10, 17803–17818. https://doi.org/10.1109/ACCESS.2022.3149955.

Lipsa, S., Dash, R. K., Ivković, N. et al. (2023) Task Scheduling in Cloud Computing: A Priority-Based Heuristic Approach. *IEEE Access*. 11, 27111–27126. https://doi.org/10.1109/ACCESS.2023.3255781.

Mell, P. M., & Grance, T. (2011) *The NIST definition of cloud computing*. National Institute of Standards and Technology. Report number: NIST SP 800-145. https://doi.org/10.6028/NIST.SP.800-145

Murad, S. A., Muzahid, A. J. M., Azmi, Z. R. M. et al. (2022) A review on job scheduling technique in cloud computing and priority rule based intelligent framework. *Journal of King Saud University - Computer and Information Sciences*, 34(6), 2309–2331. https://doi.org/10.1016/j.jksuci.2022.03.027.

Murad, S. A., Azmi, Z. R. M., Muzahid, A. J. M. et al. (2024) Priority based job scheduling technique that utilizes gaps to increase the efficiency of job distribution in cloud computing. *Sustainable Computing: Informatics and Systems*. 41, art. ID 100942. https://doi.org/10.1016/j.suscom.2023.100942.

Sanjalawe, Y., Al-E'mari, S., Fraihat, S. et al. (2025) AI-driven job scheduling in cloud computing: A comprehensive review. *Artificial Intelligence Review*. 58(7), 197. https://doi.org/10.1007/s10462-025-11208-8.

Santhosh, B., Manjaiah, D. H. & Padma Suresh, L. (2016) A survey of various scheduling algorithms in cloud environment. In: *2016* International *Conference on Emerging Technological Trends (ICETT)*, *21-22 October 2016, Kollam, India*. New York, USA, IEEE. https://doi.org/10.1109/ICETT.2016.7873782.

Silva Filho, M. C., Oliveira, R. L., Monteiro, C. C. et al. (2017) CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, *8-12 May 2017, Lisbon, Portugal*. New York, USA, IEEE. pp. 400–406.

Sridhar, M. & Babu, G. R. M. (2015). Hybrid Particle Swarm Optimization scheduling for cloud computing. In: *2015 IEEE* International *Advance Computing Conference (IACC)*, *12-13 June 2015, Bangalore, India*. New York, USA, IEEE. pp. 1196–1200.

Sutar, S., Byranahallieraiah, M. & Shivashankaraiah, K. (2024) A Dual-Objective Approach for Allocation of Virtual Machine with improved Job Scheduling in Cloud Computing. *The International Arab Journal of Information Technology*. 21(1), 46-56. https://doi.org/10.34028/iajit/21/1/4.

Wei, Y., Pan, L., Liu, S. et al. (2018) DRL-Scheduling: An Intelligent QoS-Aware Job Scheduling Framework for Applications in Clouds. *IEEE Access*. *6*, 55112–55125. https://doi.org/10.1109/ACCESS.2018.2872674.

Zhu, Q.-H., Tang, H., Huang, J.-J. et al. (2021) Task Scheduling for Multi-Cloud Computing Subject to Security and Reliability Constraints. *IEEE/CAA Journal of Automatica Sinica*. 8(4), 848–865. https://doi.org/10.1109/JAS.2021.1003934.