11 (2) 2025 106-122 ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
Register: *Jurnal Ilmiah Teknologi Sistem Informasi*

Contents lists available at www.journal.unipdu.ac.id

Register

Journal Page is available to www.journal.unipdu.ac.id/index.php/register

Research article

ECO-FISH: An Enhanced Cloud Task Scheduling Using Opposition-Based Artificial Fish Swarm Algorithm

Ary Mazharuddin Shiddiqi[a], Henning Titi Ciptaningtyas[b], Jonathan Leonardo[c], Fayruz Rahma[d]*

[a, c] *Department of Informatics, Institut Teknologi Sepuluh Nopember, Jln Teknik Kimia, Kampus ITS Sukolilo, Surabaya, 60111, East Java, Indonesia*

[b] *Department of Information Technology, Institut Teknologi Sepuluh Nopember, Jln Teknik Kimia, Kampus ITS Sukolilo, Surabaya, 60111, East Java, Indonesia*

[d] *Department of Informatics, Universitas Islam Indonesia, Jln. Kaliurang Km 14.5, Daerah12 Istimewa Yogyakarta, Indonesia*

email:[a] ary.shiddiqi@its.ac.id, [b]henning@its.ac.id, [c]jonathanleonardo123@gmail.com, [d]fayruz.rahma@uii.ac.id*

** Correspondence*

ARTICLE INFO

Article history:
Received 8 January 2025
Revised 25 June 2025
Accepted 27 December 2025
Available online 28 December 2025

Keywords:
Cloud Provisioning; Task Scheduling; Artificial Intelligence; Fish Swarm Algorithm; Opposition-Based Learning.

Please cite this article in IEEE style as:
A. M. Shiddiqi, H. T. Ciptaningtyas, J. Leonardo, F. Rahma, "ECO-FISH: An Enhanced Cloud Task Scheduling Using Opposition-Based Artificial Fish Swarm Algorithm," *Register: Jurnal Ilmiah Teknologi Sistem Informasi*, vol. 11, no. 2, pp. 106-122, 2025.

ABSTRACT

The rapid expansion of cloud computing has increased the complexity of task scheduling and resource management across heterogeneous and dynamic environments. Conventional heuristic methods often converge prematurely and lead to imbalanced virtual machine (VM) utilization. To address these challenges, this study introduces ECO-FISH, a hybrid Opposition-Based Artificial Fish Swarm Algorithm (AFSA) designed for efficient cloud task scheduling. AFSA is chosen for its swarm intelligence behaviors—prey, follow, and swarm—that enable effective local exploration with low computational cost, while Opposition-Based Learning (OBL) is integrated to strengthen global exploration by evaluating opposite task–VM mappings, helping the algorithm escape local optima and maintain population diversity. This synergy improves the balance between exploration and exploitation while retaining algorithmic simplicity. The proposed ECO-FISH is implemented in CloudSim and benchmarked against GA, PSO, and the baseline AFSA using three workload distributions: uniform, normal, and stratified. Experimental results show that AFSA alone reduces makespan by 28–45%, increases throughput by 34–84.9%, and improves utilization by 44.12–64.59% compared to GA. The OBL enhancement in ECO-FISH provides additional gains of up to 1.6%, showing the most significant improvement under heterogeneous, stratified workloads with high variance. Overall, AFSA performs well on uniform datasets, while ECO-FISH (AFSA + OBL) offers superior adaptability and stability in variable cloud environments.

1. Introduction

A well-managed cloud resource management system enables providers to optimize resources without violating established service-level agreements (SLAs). Cloud resource management encompasses various aspects, including provisioning, scheduling, and monitoring existing resources, as well as mapping user workloads to appropriate resources based on predefined quality of service (QoS) requirements. Once cloud resource provisioning is implemented, a task scheduling system processes incoming workloads according to specified deadlines or budgets. The purpose of task scheduling is to minimize task execution time and maximize the utilization of available resources [1][2]. The efficient allocation and scheduling of tasks directly affect the overall efficiency and quality of cloud computing services [3]. Thus, task scheduling algorithms have become a critical issue in cloud computing environments. Despite extensive research efforts, achieving optimal task scheduling that adapts to the dynamic and heterogeneous nature of cloud environments remains a persistent challenge, particularly

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

in relation to avoiding local optima and improving convergence speed-gaps that this work aims to address.

Researchers have explored various techniques, including heuristic, metaheuristic, and hybrid approaches, to address cloud task scheduling. For instance, evolutionary algorithms such as the genetic algorithm (GA)[4][5] and particle swarm optimization (PSO) [6][7] have been extensively used due to their ability to balance loads and reduce execution time. Moreover, task scheduling frameworks that leverage artificial intelligence and machine learning have demonstrated promising results in predicting workload patterns and adapting to dynamic cloud environments [8][9][10]. However, existing techniques exhibit several critical drawbacks. Studies indicate that although heuristic and metaheuristic algorithms attempt to balance exploration and exploitation, they often struggle with persistent issues, including becoming trapped in local optima and achieving low convergence speeds. These limitations are particularly evident in large-scale cloud environments [13][14]. Hybrid models that combine multiple algorithms aim to mitigate these issues but still fall short in certain scenarios [15]. Furthermore, the rapid scalability and dynamic nature of cloud resources complicate the task scheduling process, necessitating more robust and adaptive algorithms [16]. Despite these advancements, challenges remain in achieving optimal task-scheduling solutions capable of adapting to the ever-changing demands of cloud computing [11][12].

To address these persistent limitations, this research proposes a task scheduling algorithm for cloud computing environments based on the Artificial Fish Swarm Algorithm (AFSA), inspired by the behavior of fish schools searching for food sources [17]. The key innovation lies in integrating the opposition-based learning (OBL) algorithm with AFSA to systematically overcome the local optima trap. Unlike existing AFSA implementations that rely on generic diversity boosters such as random restarts, our approach incorporates OBL to consider the opposite values of the original estimates, providing a low-cost and bounded mechanism for discrete task-to-VM assignments. This approach differs from previous methods in three fundamental ways. First, it introduces a structured counterproposal generator that preserves AFSA's computational simplicity while expanding the search space. Second, it implements discrete OBL encoding specifically designed for combinatorial task-to-VM mappings, thereby extending OBL's practical applicability beyond continuous optimization vectors. Third, it couples AFSA's local search operators with opposition sampling focused on overloaded and underloaded VMs, thereby reducing VM hotspots and long-tail completion times commonly encountered in heterogeneous workloads. Experiments were conducted using the CloudSim simulation tool [18]. We observed the makespan, average execution time, throughput, resource utilization, and energy consumption of the proposed algorithm and compared its performance with that of other established algorithms.

Scheduling in a cloud environment is an NP-hard optimization problem [19][20]. Depending on the cloud infrastructure and user requests, the system can experience varying loads, such as underloaded, overloaded, or balanced conditions. Task scheduling methods in cloud computing are generally categorized into heuristic, metaheuristic, and hybrid approaches [21]. Several benchmark parameters are commonly used to measure the optimization level of these algorithms, including the makespan, throughput, communication time, average execution time, resource utilization, and total energy consumption [22][23]. These parameters help evaluate the effectiveness and efficiency of different task scheduling techniques in cloud environments, ensuring optimal resource allocation and improved performance.

Multidimensional optimization problems are central to cloud task scheduling, where multiple variables such as resource allocation, task priority, and execution time must be optimized simultaneously [24]. A novel approach using AFSA [12] was proposed for solving multidimensional 0-1 knapsack problems. The binary approach integrates the basic behaviours of artificial fish (chasing, swarming, searching) into crossover and mutation operations similar to those in traditional binary algorithms. Research in [25] solved a multidimensional knapsack problem (MKP) by improving AFSA and proposed Computation Scheduling Based on the Artificial Fish Swarm Algorithm (CS-AFSA) for optimal scheduling. In this approach, a scheduling scheme is encoded as an artificial fish, with delay as the optimization objective, and the optimal solution is obtained through the behavior of the artificial fish swarm.

 http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

The advantages of an algorithm can be maximized by combining it with others in a hybrid model. Rawat et al. [26] developed a hybrid genetic algorithm-artificial neural network (GA-ANN) algorithm for optimizing virtual machine allocation and task scheduling. The study used a self-generated dataset alongside the San Diego Supercomputer Center (SDSC) Blue Horizon logs to simulate real-world scenarios. Various control variables, including the number of tasks, virtual machines, data centres, bandwidth, CPU, and RAM, were used into the cloud computing simulations, providing a comprehensive reference for dataset types and system configurations.

An optimization technique that simultaneously considers opposite solutions can enhance machine learning and computational algorithms. Mahdavi et al. [27] stated that the OBL algorithm has significant potential for solving diverse scientific and technological problems. In dynamic cloud runtime environments, premature convergence, entrapment in local optima, and the imbalance between exploration and exploitation remain challenging issues. The Gabor filter and OBL methods were applied in the multi-verse optimizer (MVO) algorithm (referred to as the GOMVO algorithm) to solve these issues [18]. GOMVO enhances the optimization process and improves the performance and robustness of cloud task scheduling.

Despite significant advancements in heuristic and metaheuristic scheduling, many approaches exhibit instability under heterogeneous and time-varying workloads. A persistent limitation is premature convergence, which leads to VM hotspots, resource imbalance, and long-tail task completion times. Although the Artificial Fish Swarm Algorithm (AFSA) provides efficient local refinement, it often becomes trapped in local optima when the objective landscape is highly irregular. Existing diversity boosters, such as random restarts or mutation-based perturbations, can temporarily enhance exploration but introduce non-trivial computational overhead and lack structural consistency. Therefore, there remains a crucial need for a low-cost, structured exploration strategy that maintains AFSA's simplicissty while continuously introducing adaptive diversity to prevent premature convergence and enhance robustness in dynamic cloud environments.

ECO-FISH augments AFSA with Opposition-Based Learning (OBL) as a bounded, integer-safe counterproposal generator for discrete task to VM assignments. For a candidate mapping x with per-gene bounds [L,U], ECO-FISH constructs its opposite x' = L + U - x (discretized to valid VM IDs) and keeps the better of $\{x, x'\}$. This structured form of diversity expands the search space with negligible computational overhead, enabling AFSA to escape local minima while preserving its efficient prey, follow, swarm improvements.

The contributions of this research are as follows: (1) A hybrid AFSA+OBL scheduler that systematically injects bounded "opposite" schedules to avoid premature convergence while preserving AFSA's operators and runtime simplicity, (2) A discrete OBL encoding for combinatorial task-to-VM genes, including feasibility handling and tie-breaking, making OBL practical beyond continuous optimisation vectors, (3) A balancing-aware search mechanism that reduces VM hot-spots and long-tail completion times by coupling AFSA's local moves with opposition sampling on overloaded and underloaded VMs, and (4) A comprehensive evaluation against GA, PSO, and AFSA on heterogeneous workloads using standard metrics within a CloudSim setting.

2. Materials and Methods

2.1 Problem Definition

We examined the non-preemptive scheduling of a set of independent tasks J={1,…,n} onto a heterogeneous pool of virtual machines (VMs) V={1,…,m}. Each task i has an estimated processing requirement w_i (e.g., MI or seconds on a reference CPU). Each VM v has a processing speed s_v and capacity constraints (CPU/RAM). The processing time of i on v is:

$$p_{iv} = \frac{w_i}{s_v} \quad (1)$$

Decision variables $x_{iv} \in \{0,1\}$ indicate assignment (1 if task i runs on VM v, else 0), subject to constraint:

$$\sum_{v=1}^{m} x_i = 1 ; \quad (2)$$

where $\forall I \in J$ (each task assigned exactly once) and Feasibility $(i,v) = 1 \Rightarrow x_{iv} = 1$ only if resource demands of i fit VM v. The primary objective is to minimize makespan:

$$C_{max}(x) = max_{e \in V} \sum_{i=1}^{n} p_{iv} x_{iv} \quad (3)$$

http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

2.2 Background

2.2.1 Artificial Fish Swarm Algorithm (AFSA)

AFSA moves each artificial fish according to its current state and the surrounding environment at a future time [28]. The environment is influenced by the behavior of the artificial fish itself and the states of neighboring fish. The subsequent movement of an artificial fish is determined by its visual range and crowd factor, based on Equation 4:

$$v = \delta \times n \quad (4)$$

where v is the visual value of the artificial fish, δ is a predefined visual constant ($0 < \delta < 1$), and n is the maximum allowable Hamming distance between two fish. Hamming distance is used because Euclidean distance does not accurately measure the distance between artificial fish whose positions are represented by 0 or 1. Since the position of the fish is not only represented by the bits 0 and 1, thus the Euclidean distance can be calculated. Therefore, the visual value in the AFSA is represented using Equation 5:

$$v = \delta \times \text{maxEuclideanDistance} \quad (5)$$

Once the visual value is obtained, the crowd factor (C_f) of an artificial fish can be represented using Equation 6:

$$C_f = \frac{np^i}{N} \quad (6)$$

where np^i represents the number of fish within the visual scope of fish i, and N represents the total fish population. There are five behavioral options that an artificial fish can perform in b-AFSA. First, chasing (Following) Behavior: If fish X_i has neighbors within its visual range, the best neighbor position of X_i can be determined (X_{best}). If the fitness value of X_{best} is better than that of X_i ($Y_{best} > Y_i$), and its position is not too crowded, the chasing behavior will be executed. In b-AFSA, chasing behavior is implemented by performing a crossover between fish X_i and fish X_{best}. Second, swarming Behavior: If the fitness value of X_{best} is less than or equal to that of fish X_i ($Y_{best} \leq Y_i$), one of the neighbors of fish X_i is randomly chosen (X_{rand}). If the fitness value of X_{rand} is better than that of fish X_i ($Y_{rand} > Y_i$), the swarming behavior will be executed. In b-AFSA, swarming behavior is implemented by performing a one-position mutation on fish X_i. Third, Searching (Preying) Behavior: There are two situations in which the searching behavior will be executed: (a) when the position is not too crowded, but X_{best} and X_{rand} do not have a fitness value better than X_i ; (b) when the position is too crowded. In these cases, one of the neighbors of fish X_i is randomly chosen again (X_{rand}). If the fitness value of X_{rand} is better than that of fish X_i ($Y_{rand} > Y_i$), the searching behavior will be executed. If not, the random behavior will be executed. The searching behavior is implemented by performing a crossover between fish X_i and fish X_{rand}. Fourth, Random Behavior: When fish X_i has no neighbors or other behaviors are not executed, fish X_i will perform random behavior. This behavior is implemented by randomly assigning position bits throughout the fish's dimensions. Fifth, Leaping Behavior: This behavior is executed every few iterations to prevent the fish population from getting trapped in a local optimum. The leaping behavior in b-AFSA is implemented by randomly selecting an artificial fish from the existing population (X_{rand}). Then, mutation is performed on X_{rand} with a probability of p_m (0.01).

The b-AFSA uses mutation and one-position crossover operations to determine the next position of each artificial fish. Crossover is used to implement the chasing and searching behaviours. In this research, two types of crossovers operations are applied: one-position crossover and uniform crossover. The results of these two crossover operations are compared, and the position with the higher fitness value is selected as the next position of the artificial fish. First, one-Position Crossover: A random index $r1$ is first selected from the position dimension of the fish n. The bits from $r1$ to n of the two selected artificial fish are then exchanged to create two possible fish positions. The position with the higher fitness value is selected as the result. For example, consider $r1 = 10$ (Table 1).

Table 1. One-Position Crossover

Point 1	1	0	0	0	1	1	0	1	0	**1**	**1**	**0**	**1**	**1**	**0**
Point 2	0	0	1	1	0	1	0	0	1	**0**	**0**	**0**	**1**	**0**	**1**
Candidate 1	1	0	0	0	1	1	0	1	0	**0**	**0**	**0**	**1**	**0**	**1**
Candidate 2	0	0	1	1	0	1	0	0	1	**1**	**1**	**0**	**1**	**1**	**1**

http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

Second, uniform crossover: Two positions of the selected fish are combined to generate a single candidate position. The bit selected at each index is determined using a random probability value of τ_1. If $\tau_1 \sim U[0, 1]$, which yields a value of less than or equal to 0.5, the bit is taken from fish 1; otherwise, it is taken from fish 2. For example, consider indices 1, 2, 6, 7, 8, 9, 11, 13 ($\tau_1 \leq 0.5$) and 3, 4, 5, 10, 12, 14, 15 ($\tau_1 > 0.5$) (Table 2).

Table 2. Uniform Crossover

Point 1	1	0	0	0	1	1	0	1	0	1	1	0	1	1	0
Point 2	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
Candidate 1	1	0	1	1	0	1	0	1	0	0	1	0	1	0	1

The mutation operation implements the swarming and leaping behaviors, where the position value at the selected index is changed to its opposite value. This opposite value is computed using a formula similar to the position calculation in OBL. The mutation process consists of a one-position mutation representing the swarming behavior, and a mutation with a probability p_m representing the leaping behavior. First, One-Position Mutation: An index value (r_2) is randomly selected, where r_2 represents represent a specific position index of an artificial fish ($r_2 \in \{1, 2, \ldots, n\}$). The value at index r_2 is then inverted, resulting in the opposite value. For example, consider $r_2 = 4$ (Table 3).

Table 3. One-Position Mutation

Point 1	1	0	0	0	1	1	0	1	0	1	1	0	1	1	0
Candidate 1	1	0	0	1	1	1	0	1	0	1	1	0	1	1	0

Second, mutation with a Probability p_m: A random probability value τ_2 will be used to determine whether the value at index j ($j = 1, 2, \ldots, n$, where n is the maximum index of the fish's position) will be inverted or not. If $\tau_2 \sim U[0, 1]$ yields a value less than or equal to p_m (generally, $p_m = 0.01$ is used), then the value at index j will be inverted, resulting in the opposite value. For example, the indices 4 and 14 yield the random values $\tau_2 \leq 0.01$ (Table 4).

Table 4. Mutation with a Probability

Point 1	1	0	0	0	1	1	0	1	0	1	1	0	1	1	0
Candidate 1	1	0	0	1	1	1	0	1	0	1	1	0	1	0	0

Once the candidate position of a certain fish is obtained, a selection process is carried out to determine whether the current position should be replaced. This process compares the fitness value of the current position with that of the candidate position. If the candidate position has a higher fitness value, it replaces the current position; otherwise, the fish is assumed not to move and maintains its current position (Equation 7) [29]:

$$x_i^{(t+1)} \begin{cases} y_i^{(t+1)}, if\ z\left(y_i^{(t+1)}\right) > z\left(x_i^{(t)}\right), i = 1,2,\ldots,N \\ x_i^{(t)} \end{cases} \quad (7)$$

where $x_i^{(t+1)}$ is the next position of the fish, $x_i^{(t)}$ is the current position, $y_i^{(t+1)}$ is the candidate position, and z denotes the fitness evaluation function.

In several cases, another version of b-AFSA, the Simplified Binary Artificial Fish Swarm Algorithm (s-bAFSA) is to further prevent solution points from getting trapped in local optima. This variant employs a more advanced leaping method, based on reinitializing the entire artificial fish population to diversify the search and avoid trapping the solution points in non-optimal positions [30]. Hence, the leaping method in b-ASFA derives from mutation process with probability p_m, whereas in b-AFSA it involves reinitialization of the entire population of artificial fish in s-bAFSA.

2.2.2 Opposition-based learning (OBL)

OBL compares the current estimated value with its opposite to identify the optimal solution by simultaneously considering an estimate value and its opposite. This approach can improve the performance of existing algorithms [27]. In many cases, the initial population in intelligent algorithms is generally started at random positions. These points then begin to search and move toward potential solution points. Logically, it is necessary to explore all directions simultaneously, or more precisely, to also consider the direction opposite to the current position. If we are searching for a solution at x, then calculating its opposite value, $\tilde{x}$ is the first logical step. The definition of the opposite value in OBL can be expressed using the following equation:

$$\tilde{x} = a + b - x \quad (8)$$

 http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

where $\tilde{x}$ is the opposite value of x, with $x \in$ R, and x has the interval $x \in [a, b]$. For $a = 0$ and $b = 1$, the opposite value $\tilde{x}$ becomes

$$\tilde{x} = 1 - x \quad (9)$$

By the same analogy, the opposite value in a multi-dimensional case can be defined using the following equation:

$$\tilde{x}_i = a_i + b_i - x_i \quad for\ i = 1, \ldots, n \quad (10)$$

where P($x_1, x_2, \ldots, x_n$) is a point in an n-dimensional coordinate system, with $x_1, \ldots, x_n \in$ R and $x_i \in [a_i, b_i]$.

Thus, the concept of OBL can be explained as follows: Suppose *f(x)* is the function of interest and *g(.)* is the feasibility evaluation function. If $x \in [a, b]$ is an initial position (chosen randomly) and ˜x is its opposite value, then in each iteration the values *f(x)* and $f(\tilde{x})$ will be calculated. The exploration and search process will continue with x if $g(f(x)) > g(f(\tilde{x}))$; otherwise, the search will proceed with $\tilde{x}$. Depending on which value is closer to the optimal solution, the search space interval can be continuously divided into two parts until one of these points is sufficiently close to the desired solution. Table 5 is an example of OBL implementation for the position of artificial fish.

Table 5. Opposition-based Learning

Point 1	0	2	4	6	1	3	8	5	7
Opposite 1	8	6	4	2	7	5	0	3	1

A fish's position value represents the ID of a virtual machine in a data center. The opposite of the value 0 is 8 because, in the test system design created by the author, the data center contains nine virtual machines with IDs ranging from 0 to 8. Therefore, each index value in the fish's position can only be represented by the numbers 0 through 8. Thus, 0 is the opposite of 8, 1 is the opposite of 7, and so on.

2.3 Methods

The proposed ECO-FISH method enhances AFSA with OBL to enhance cloud task scheduling. The AFSA component mimics the behavior of fish swarms to efficiently explore and exploit the solution space. The integration of OBL strengthens the search process by considering both current and opposite positions, thereby increasing the likelihood of escaping local optima. The objective is to optimize resource allocation, reduce makespan, and improve the overall scheduling performance in cloud environments. The scheduling algorithm is integrated within the data center broker, which manages the allocation of user tasks to suitable virtual machines (VMs) across multiple data centers (Figure 1).

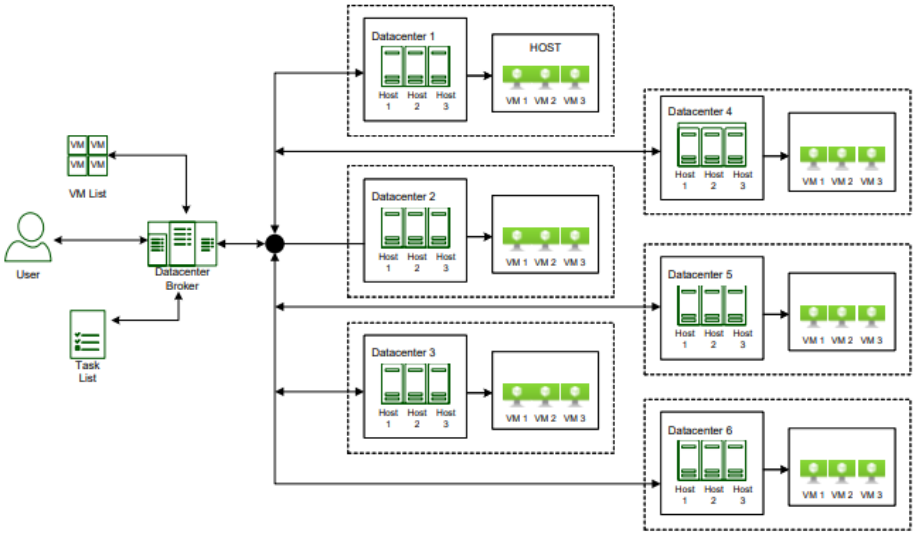

Fig. 1. The proposed system architecture implemented in the CloudSim simulation system. The scheduling algorithm is implemented in the data center broker, which manages the distribution of user tasks to the appropriate virtual machines (VMs) across various data centers.

2.3.1 The ECO-FISH Algorithm

Our proposed ECO-FISH (Algorithm 1) forms by generating an initial population of artificial fish with random positions. Each value in a fish's position represents the ID of the virtual machine to which a task will be assigned. After forming the initial population, the fitness value of each fish is calculated, and the artificial fish with the best fitness value is identified. The first iteration of AFSA then begins. In

http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

each iteration, the behavior performed by each fish is determined. To do this, we first identify all the neighbouring artificial fish and calculate their crowding factors. Once the neighbor data and crowding factor values are obtained, the behavior-determination process can proceed. During the experiments, we observe the makespan value and record the smallest values achieved, which are regarded as the good results.

Then, the position of the selected movement result of an artificial fish ($X'_{proposed}$) is evaluated against its current position. If the fitness value of the fish's current position is already better (smaller) than that of the proposed movement result, the fish does not move. Conversely, if the proposed position has a better fitness value, the fish's position and fitness value are updated accordingly. At this stage, the opposite value of the selected movement result position ($X'_{proposed}$) is also considered when evaluating the final position of the fish, including chasing, swarming, preying, random, or leaping behavior (Table 6). After the position evaluation, we determine whether the fish's current position represents the best global fitness value. If so, the fish is labeled as *bestfish*. The movement determination and position evaluation processes continue for each fish in an iteration. The leaping process is executed when the current iteration is a multiple of the *leapingFreq* value. Once all AFSA iterations are completed, the best fish is obtained. The tasks in the batch are then allocated using the position values of this best fish. Finally, the CloudSim simulation is executed, and the performance metrics are obtained.

Algorithm 1 The ECO-FISH algorithm

```
Initialize: fishNum, dim, MaxIteration, delta, visualScope, leapingFreq
Initialize: Initial random position of fish population
    while iteration < MaxIteration do
        Calculate the fitness value of each fish
        Find the best fish and its fitness value (Ybestfish)
        for each fish i do
            Find all fish neighbors i and calculate Cf
            Determine and execute behavior to get Xproposed
            Calculate the value of X' proposed
            Evaluate fish position i against Xproposed and X' proposed
        end for
        If MOD(iter, leapingFreq) == 0 then
            Execute the leaping method
        end if
        if iter == MaxIteration then
            Output: Optimization evaluation parameters
            Allocate tasks to the selected virtual machine
            if all tasks have been scheduled then
                Run CloudSim simulation
                Output: Optimization evaluation parameters
            Else
                Re-run scheduling for the next tasks
                Go back to Initial random fish population
            end if
        end if
        Increment iteration
    end while
```

Table 6. Illustration of task allocation to VMs using the ECO-FISH algorithm. X denotes the allocation of tasks to virtual machines based on the value of $X_{proposed}$, while X′ represents the allocation of tasks to virtual machines based on the position value $X_{proposed}$

	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
VM_0							X′	X	
VM_1				X	X′				
VM_2	X								X′
VM_3		X′	X						
VM_4					X	X′			
VM_5		X	X′						
VM_6	X′								X
VM_7				X′	X				
VM_8							X	X′	

2.3.2 Fitness function

We used the makespan as the fitness function to measure the quality of a proposed solution generated in the experimental scenarios (Equation 8). Our objective is to minimize the time required to complete all tasks by placing them to appropriate virtual machines. In other words, the proposed system is expected to complete the given tasks as quickly as possible with minimal costs [31]. Thus, we focus on

 http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

minimizing the obtained makespan value, where makespan represents the total time needed to complete all existing tasks:

$$F = \max_{j \in vm}(\sum CT(T_{i \in task}, R_j)) \quad (11)$$

where F is the fitness function, T_i is a task executed by the selected virtual machine, R_j is the selected virtual machine used to complete task i, and CT (completion time) is the time required to complete the task.

The completion time refers to the state in which a process has been successfully executed and stopped. It is represented as the sum of the arrival time (the time required for a task to enter the system and be ready for execution), waiting time (the time a task spends in the queue before its execution starts), and execution time (the time required to complete the execution of a task) [4][32]. However, in the CloudSim simulation environment, the waiting time cannot be obtained before the completion of the simulation. Therefore, the completion time formula used in this research is presented in Equations 12–14:

$$CT = AT + ET \quad (12)$$

$$AT = \frac{Task\ Length}{VM\ Bandwidth} \quad (13)$$

$$ET = \frac{Task\ Length}{Processing\ Unit\ VM \times MIPS\ VM} \quad (14)$$

where CT is the completion time, AT is the arrival time, and ET is the execution time. Using these values, the completion time of each task can be calculated, and the total completion time for each virtual machine can be determined. As a result, the makespan value for completing the entire set of tasks can also be identified.

2.3.3 Dataset

Three datasets with different characteristics were used to evaluate the performance of the proposed ECO-FISH algorithm under various workload scenarios: the San Diego Supercomputer Center (SDSC) Blue Horizon Log, a uniform random dataset, and a stratified random dataset. These diverse datasets provide a comprehensive foundation for evaluating the performance and optimization of AFSA and its combination with OBL in different scenarios. First, San Diego Supercomputer Center (SDSC) Blue Horizon Log: The first dataset is the San Diego Supercomputer Center (SDSC) Blue Horizon Log, which contains real workload traces collected from a supercomputing environment over a two-year period [33]. The dataset includes job-level details such as the number of nodes, requested and actual CPU time, submission and start times, and job completion durations. These data provide realistic distributions of job lengths and inter-arrival times and are widely used in task scheduling research for benchmarking and validation. Second, Uniform random tasks: The second dataset was synthetically generated to represent evenly distributed workloads. Task lengths were created using a uniform random generator with lower and upper bounds of 10,000 and 40,000, respectively. Each dataset instance contained between 1,000 and 10,000 tasks. This distribution ensures equal probability for all task lengths, allowing evaluation of the algorithm's stability under balanced workload conditions. Third, Stratified random tasks: The third dataset was derived by grouping task lengths from the SDSC Blue Horizon Log into 100 classes to approximate the real-world heterogeneity observed in the original data. Each class width was set to 88,000 based on the difference between the minimum and maximum values in the SDSC dataset. The relative frequency of each class was then used to generate a smaller stratified sample containing between 1,000 and 10,000 entries. This stratified dataset retains the workload variability of the SDSC dataset while reducing computational cost, enabling scalable simulation of heterogeneous conditions for evaluating ECO-FISH.

All experiments were conducted using the CloudSim simulation framework with identical system configurations across datasets. A consistent random seed was used to ensure reproducibility, and all dataset generation scripts will be made available upon request or deposited in a public repository following publication.

2.3.4 Evaluation parameters

Evaluation parameters are used to evaluate the efficiency and effectiveness of the proposed method compared to other approaches. These parameters are applied consistently across all experimental scenarios to evaluate the performance of each algorithm.

 http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

Table 7. Evaluation parameters

Parameters	Detail	Formula	
Makespan	Measures the total time taken to complete all tasks. The lower the value, the better the performance of an algorithm	$\max_{i \in tasks}\{F_i\}$	(15)
Average Start Time	Indicates the average time a task takes to begin execution on a processor. The lower this value, the better the performance of an algorithm	$\frac{\sum_{i=1}^{n} Time\ R_i\ Start}{nR}$	(16)
Average Finish Time	Indicates the average time needed for a task to complete after being inserted into the virtual machine. The lower the value, the better the algorithm's performance	$\frac{\sum_{i=1}^{n} Time\ R_i\ Finish}{nR}$	(17)
Average Execution Time	Measures each task's time to run. Lower values indicate a better algorithm performance	$\frac{\sum_{i=1}^{n} Time\ T_i}{nR}$	(18)
Total Wait Time	Indicates the total waiting time of a task before processing begins. The lower the value, the better the algorithm performance	$\sum T_{delay}$	(19)
Scheduling Length	Determines the duration of task scheduling processes. Lower values indicate a better algorithm performance	$Scheduling\_Time + Makespan$	(20)
Throughput	Evaluates the number of tasks completed within a given timeframe. Higher values indicate a better algorithm performance	$\frac{nT}{\text{Makespan}}$	(21)
Resource Utilization	Measures the percentage of cloud resources used during the simulation. Higher values indicate a better algorithm performance	$\frac{\sum_{i=1}^{n} Time\ R_i\ Finish}{Makespan \times nR}$	(22)
Imbalance Degree	Measures load distribution across resources. Lower values indicate a better algorithm performance	$\frac{ET_{\max} - ET_{\min}}{ET_{\text{avg}}}$	(23)
Energy Consumption	Measures the energy usage of all cloud resources during the simulation. Lower values indicate a better algorithm performance	$\sum Energy\_Consumption$	(24)

3. Results And Discussion

3.1. Environment

We configured the AFSA parameters based on the research conducted in [31], which demonstrated strong effectiveness for discrete task scheduling problems in cloud environments. Their suitability was further confirmed through preliminary tuning experiments conducted for this study. The parameter configuration shown in Table 9 was designed to maintain a balance between exploration (discovering new candidate solutions) and exploitation (refining promising solutions), while also preserving computational efficiency suitable for real-time scheduling scenarios. These parameter values were also applied when implementing AFSA-OBL in the second experimental scenario. The simulation environment consisted of six data centers, eighteen hosts, and fifty-four virtual machines (VMs), as shown in Figure 1. Each data center contained three hosts (Table 8), and each host contained three VMs (Table 9). All data centers were connected to a central data broker, which managed the distribution of tasks across VMs. The broker component was linked to both the task list and the VM list entities, facilitating dynamic task scheduling and ensuring efficient workload allocation across the cloud infrastructure.

Table 8. Host spesification

Host	Storage (MB)	RAM (MB)	Bandwidth (Mbps)	Max. Power	Static Power (%)
Host 1	1,000,000	128,000	10,000	117	50
Host 2	1,000,000	128,000	10,000	117	50
Host 3	1,000,000	128,000	10,000	117	50

Table 9. Virtual machine spesification

VM	Storage (MB)	RAM (MB)	MIPS	Bandwidth (Mbps)	No Processor
VM 1	1,000	512	400	1000	1
VM 2	1,000	1,024	500	1000	1
VM 3	1,000	2,048	600	1000	1

http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

Table 10. Parameters used for AFSA and ECO-FISH

No.	AFSA Parameter	Parameter Value	Justification
1	FishNum	10	The number of artificial fish used in the AFSA algorithm. The population size (FishNum = 10) was chosen to provide adequate diversity without excessive overhead; larger populations showed only marginal improvement but significantly increased simulation time.
2	Dim	9	The search space dimension (Dim = 9) corresponds to the number of simultaneous task–VM mappings evaluated per iteration, matching the system scale used in our setup.
3	MaxIteration	10	The maximum iteration count (MaxIteration = 10) was empirically determined to guarantee convergence stability within acceptable runtime; extending beyond ten iterations produced negligible gains in solution quality.
4	Thetas (θ)	0.75	The crowdedness threshold ($\theta = 0.75$) and visual range parameter ($\delta = 0.7$) were
5	Visual Parameter (δ)	0.7	adopted from [31] as they effectively regulate fish interactions—values below these limits caused early stagnation, whereas higher values reduced swarm coordination.
6	maxDistance	14.93152	The maximum Euclidean distance (maxDistance = 14.93) was derived from normalization of the search space to maintain consistent spatial scaling among fish, ensuring balanced movement dynamics.
7	leapingFreq	5	The leaping frequency (leapingFreq = 5) was selected following [32] to introduce periodic global exploration; increasing this frequency improved diversity but at the cost of longer convergence time, while lower values risked premature convergence.

3.2. Result

We conducted experiments using the three datasets and compared the performance of ECO-FISH with two established algorithms, namely GA and AFSA, based on the evaluation parameters (Section 2.3.4).

3.2.1. Makespan

The AFSA achieved the best makespan values for the uniform random dataset (Table 11) and significantly outperformed the GA, with makespan reductions of 45.45% and 28.25% for the simple and stratified datasets, respectively.

Table 11. Comparison of f average makespan on simple and stratified datasets

Cloudlets	GA		PSO		AFSA		ECO-FISH	
	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified
1000	2.262,30	24.376,20	5.468,10	38.160,60	1.202,10	26.800,80	1.180,50	26.802,60
2000	4.303,50	64.306,50	6.894,60	55.589,10	2.244,30	36.699,90	2.241,60	36.673,80
3000	6.334,80	55.450,50	8.531,70	72.849,30	3.341,40	45.921,30	3.316,20	45.923,10
4000	8.263,50	85.202,70	9.843,00	90.219,30	4.362,00	57.363,90	4.391,70	57.372,00
5000	10.094,10	89.418,30	11.209,20	112.952,40	5.445,60	69.058,50	5.469,90	69.014,40
6000	11.850,90	124.604,70	12.973,20	113.685,00	6.420,30	78.951,30	6.533,70	79.003,50
7000	13.830,00	114.748,80	14.068,50	128.116,50	7.570,50	90.678,30	7.557,90	90.571,20
8000	15.412,20	148.514,10	15.640,80	150.314,10	8.581,20	101.361,30	8.647,80	101.138,10
9000	17.151,00	156.586,20	16.988,10	173.052,60	9.544,20	111.066,00	9.611,70	111.457,50
10000	19.248,90	171.572,10	18.984,30	175.754,40	10.614,30	124.536,30	10.795,20	124.316,70

Table 12. Comparison of average makespan on SDSC dataset

Cloudlets	GA - SDSC	PSO – SDSC	AFSA - SDSC	ECOFISH - SDSC
7395	129.390,90	133.814,40	128.870,70	**126.934,80**

Table 13. Comparison of average start time on simple and stratified datasets

Cloudlets	GA		PSO		AFSA		ECO-FISH	
	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified
1000	666,02	838,65	696,69	4.689,58	488,18	660,29	486,98	660,51
2000	1.332,68	1.910,88	1.263,26	9.039,00	978,40	1.507,53	976,54	1.507,36
3000	2.023,86	2.766,87	1.862,40	13.753,35	1.494,66	2.388,33	1.491,81	2.387,52
4000	2.717,18	3.769,20	2.457,87	18.450,41	1.990,58	3.229,09	1.988,19	3.228,60
5000	3.376,20	4.608,97	3.053,93	23.185,06	2.491,92	4.077,69	2.490,50	4.077,78
6000	3.994,00	5.610,48	3.675,27	27.861,28	2.987,57	4.909,60	2.983,91	4.909,89
7000	4.692,31	6.456,61	4.283,58	32.806,05	3.508,85	5.747,85	3.503,05	5.747,20
8000	5.371,16	7.625,49	4.908,94	36.871,22	4.002,49	6.796,94	3.994,65	6.796,49
9000	6.024,99	8.460,36	5.512,12	42.493,28	4.485,72	7.574,62	4.479,65	7.575,31
10000	6.718,76	9.284,49	6.119,18	47.092,59	4.999,73	8.349,18	4.993,84	8.348,43

http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

On the other hand, ECO-FISH produced the best results for the stratified random dataset, improving performance by 0.02% compared to the original AFSA. In the SDSC dataset (Figure 12), the AFSA's makespan was 0.4% better than the GA's; however, both were outperformed by ECO-FISH, whose makespan was 1.5% lower than the AFSA's and 1.9% lower than the GA's.

3.2.2. Average Start Time

The ECO-FISH algorithm achieves the best average start time for both the simple and stratified random datasets (Table 14). The AFSA algorithm shows average start times that are 25.71% and 11.86% faster than those of the GA for the uniform and stratified random datasets, respectively. ECO-FISH further improves the average start time by an additional 0.15% and 0.02% over AFSA on these datasets. For the SDSC dataset (Table 16), AFSA achieves a better average start time, reducing it by 24.48% compared to the GA. However, this improvement is not found when comparing ECO-FISH compared to the native AFSA. Despite the small differences between AFSA and ECO-FISH, both consistently outperform the GA in terms of average start time across all three datasets.

Table 14. Final results of average start time on SDSC dataset

Cloudlets	GA - SDSC	PSO - SDSC	AFSA - SDSC	ECOFISH - SDSC
7395	28.737,85	26.175,75	**21.701,32**	21.713,78

Table 15. Average final results of the average finish time parameter for uniform and stratified datasets

Cloudlets	GA		PSO		AFSA		ECO-FISH	
	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified
1000	727,58	1.325,83	763,69	5.193,93	543,70	1.164,50	542,41	1.164,73
2000	1.394,08	2.376,32	1.329,88	9.545,69	1.032,81	2.011,61	1.030,83	2.011,40
3000	2.085,20	3.247,37	1.929,13	14.260,97	1.549,29	2.894,33	1.546,34	2.893,44
4000	2.778,09	4.238,23	2.524,13	18.954,33	2.044,85	3.734,25	2.042,40	3.733,68
5000	3.437,17	5.084,57	3.120,10	23.691,05	2.546,42	4.583,18	2.544,97	4.583,27
6000	4.055,24	6.078,03	3.741,50	28.366,32	3.041,92	5.413,64	3.038,19	5.413,97
7000	4.753,51	6.928,69	4.349,87	33.312,02	3.563,46	6.252,45	3.557,57	6.251,73
8000	5.432,42	8.094,44	4.975,35	37.378,75	4.056,93	7.303,73	4.048,98	7.303,18
9000	6.086,27	8.930,82	5.578,47	42.999,51	4.539,77	8.079,93	4.533,63	8.080,59
10000	6.779,94	9.752,07	6.185,45	47.599,39	5.054,17	8.853,72	5.048,21	8.852,89

Table 16. Average final results of the average finish time parameter for the SDSC dataset

Cloudlets	GA - SDSC	PSO - SDSC	AFSA - SDSC	ECOFISH - SDSC
7395	29,076.99	26,555.3652	**22,035.83**	22,048.99

3.2.3. Average Finish Time

The AFSA shows an average finish time that is 25.45% faster for the uniform dataset and 10.28% faster for the stratified dataset compared to the GA. ECO-FISH further improves performance by reducing the finish time by an additional 0.14% and 0.02%, respectively. Thus, ECO-FISH achieves the fastest average finish time on both the simple and stratified datasets (Table 14). However, ECO-FISH does not produce better results than the native AFSA for the SDSC dataset (Table 15). Its performance is comparable to AFSA and still achieves the best average finish time overall, reducing it by 24.22% compared to the GA.

3.2.4. Average Execution Time

The AFSA is 10.96% faster than the GA for the uniform dataset. ECO-FISH further improves execution time by approximately 0.15% for the uniform dataset. Hence, ECO-FISH provides the fastest average execution time for the uniform random dataset among the three algorithms (Table 17).

Table 17. Average final results of the average execution time parameter for uniform and stratified datasets

Cloudlets	GA		PSO		AFSA		ECO-FISH	
	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified
1000	61,56	487,18	67,00	504,35	55,51	504,21	55,43	504,22
2000	61,40	465,44	66,63	506,68	54,41	504,09	54,29	504,03
3000	61,34	480,49	66,73	507,62	54,63	506,00	54,54	505,92
4000	60,91	469,03	66,26	503,92	54,27	505,17	54,21	505,08
5000	60,97	475,60	66,16	505,99	54,51	505,49	54,47	505,50
6000	61,24	467,54	66,23	505,04	54,35	504,05	54,28	504,08
7000	61,20	472,07	66,28	505,96	54,61	504,60	54,51	504,53
8000	61,26	468,95	66,40	507,53	54,44	506,75	54,33	506,69
9000	61,28	470,46	66,34	506,23	53,98	505,31	53,98	505,28
10000	61,17	467,59	66,27	506,80	54,44	504,54	54,37	504,46

http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

However, for the stratified dataset, the GA outperforms both AFSA and ECO-FISH, achieving an execution time that is 6.45% faster than AFSA. ECO-FISH improves upon the AFSA's execution time by approximately 0.01% for this dataset. For the SDSC dataset, the AFSA achieves the best average execution time, reducing it by 1.37% compared to the GA (Table 18). ECO-FISH does not improve upon the AFSA's average execution time in this case.

Table 18. Average final results of the average execution time parameter for the SDSC dataset

Cloudlets	GA - SDSC	PSO - SDSC	AFSA - SDSC	ECOFISH - SDSC
7395	339,14	379,61	**334,51**	335,20

3.2.5. Total Wait Time

ECO-FISH achieved the lowest total wait time among all algorithms for the simple and stratified datasets (Table 19). The AFSA outperformed the GA by 25.6% and 28.23% on the simple and stratified datasets, respectively, with ECO-FISH further improving the performance by 0.14% and 0.003% compared to the AFSA. For the SDSC dataset (Table 20), the AFSA reduced the wait time by 24.49% compared to the GA, but ECO-FISH did not improve upon the AFSA's performance.

Table 19. Average final results of the total wait time parameter for uniform and stratified datasets

Cloudlets	GA		PSO		AFSA		ECO-FISH	
	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified
1000	665.416,80	838.053,90	696.094,20	4.688.979,30	487.582,20	659.693,70	486.378,00	659.905,20
2000	2.664.161,10	3.820.557,60	2.525.311,80	18.076.808,70	1.955.607,30	3.013.850,70	1.951.887,60	3.013.528,50
3000	6.069.771,00	8.298.824,40	5.585.400,00	41.258.262,60	4.482.182,70	7.163.199,90	4.473.621,00	7.160.769,90
4000	10.866.312,90	15.074.415,00	9.829.074,60	73.799.243,10	7.959.921,30	12.913.941,60	7.950.360,60	12.911.980,50
5000	16.877.987,10	23.041.872,00	15.266.657,70	115.922.301,30	12.456.580,50	20.385.469,80	12.449.505,60	20.385.879,30
6000	23.960.410,20	33.659.296,20	22.048.012,80	167.164.074,90	17.921.809,80	29.453.972,40	17.899.834,50	29.455.765,20
7000	32.841.938,70	45.192.099,60	29.980.876,50	229.638.170,70	24.557.727,60	40.230.721,80	24.517.179,00	40.226.210,10
8000	42.964.488,00	60.999.116,40	39.266.757,00	294.964.987,50	32.015.137,50	54.371.055,60	31.952.416,50	54.367.141,50
9000	54.219.517,20	76.137.869,70	49.603.710,60	382.434.092,10	40.366.081,80	68.166.208,80	40.311.467,10	68.172.399,00
10000	67.181.649,30	92.838.858,30	61.185.838,50	470.919.872,70	49.991.277,60	83.485.801,90	49.932.370,80	83.478.268,80

Table 20. Average final results of the total wait time parameter for the SDSC dataset

Cloudlets	GA - SDSC	PSO – SDSC	AFSA - SDSC	ECOFISH - SDSC
7395	212.511.975,30	193.565.257,20	**160.476.804,00**	160.568.985,60

3.2.6. Scheduling Length

ECO-FISH achieves the shortest scheduling length for the simple and stratified datasets (Table 21). The AFSA reduces the scheduling length by 25.61% and 11.18% compared to the GA on the simple and stratified datasets, respectively, while ECO-FISH shortens the AFSA's scheduling length by 0.14% and 0.00003%. For the SDSC dataset (Table 22), the AFSA reduces the scheduling length by 24.47% compared to the GA. ECO-FISH requires a slightly extended longer scheduling length; however, it remains faster than the GA.

Table 21. Average final results of the scheduling length parameter for uniform and stratified datasets

Cloudlets	GA		PSO		AFSA		ECO-FISH	
	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified
1000	667.679,10	862.430,10	701.562,30	4.727.139,90	488.784,30	686.494,50	487.558,50	686.707,80
2000	2.668.464,60	3.884.864,10	2.532.206,40	18.132.397,80	1.957.851,60	3.050.550,60	1.954.129,20	3.050.202,30
3000	6.076.105,80	8.354.274,90	5.593.931,70	41.331.111,90	4.485.524,10	7.209.121,20	4.476.937,20	7.206.693,00
4000	10.874.576,40	15.159.617,70	9.838.917,60	73.889.462,40	7.964.283,30	12.971.305,50	7.954.752,30	12.969.352,50
5000	16.888.081,20	23.131.290,30	15.277.866,90	116.035.253,70	12.462.026,10	20.454.528,30	12.454.975,50	20.454.893,70
6000	23.972.261,10	33.783.909,00	22.060.986,00	167.277.759,90	17.928.230,10	29.532.923,70	17.906.368,20	29.534.768,70
7000	32.855.768,70	45.306.848,40	29.994.945,00	229.766.287,20	24.565.298,10	40.321.400,10	24.524.736,90	40.316.781,30
8000	42.979.900,60	61.147.630,50	39.282.397,80	295.115.301,60	32.023.718,70	54.472.416,90	31.961.064,30	54.468.279,60
9000	54.236.668,20	76.294.455,90	49.620.698,70	382.607.144,70	40.375.626,00	68.277.274,80	40.321.078,80	68.283.856,50
10000	67.200.898,20	93.010.430,40	61.204.822,80	471.095.627,10	50.001.891,90	83.610.338,10	49.943.166,00	83.602.585,50

Table 22. Average final results of the scheduling length parameter for the SDSC dataset

Cloudlets	GA - SDSC	PSO - SDSC	AFSA - SDSC	ECOFISH - SDSC
7395	212.641.366,20	193.699.071,60	**160.605.674,70**	160.695.920,40

3.2.7. Throughput

The AFSA achieved the highest throughput on the uniform random dataset, while ECO-FISH performed best on the stratified random dataset (Table 23). The AFSA improved throughput by 84.9% over the GA on the uniform dataset and by 34.77% on the stratified dataset. ECO-FISH further increased

http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

throughput on the stratified dataset by 0.03% but did not improve performance on the uniform dataset. For the SDSC dataset (Table 24), ECO-FISH achieved the highest throughput. The AFSA improved throughput by 0.7% over the GA, and ECO-FISH further increased throughput by 1.39% compared to the AFSA.

Table 23. Average final results of the throughput parameter for uniform and stratified datasets

Cloudlets	GA		PSO		AFSA		ECO-FISH	
	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified
1000	0,443	0,041	0,180	0,030	0,832	0,037	0,848	0,037
2000	0,466	0,033	0,290	0,040	0,891	0,055	0,892	0,055
3000	0,474	0,055	0,350	0,040	0,898	0,065	0,905	0,065
4000	0,484	0,047	0,410	0,050	0,917	0,070	0,911	0,070
5000	0,496	0,057	0,450	0,050	0,918	0,072	0,914	0,072
6000	0,507	0,048	0,460	0,050	0,935	0,076	0,918	0,076
7000	0,507	0,062	0,500	0,060	0,925	0,077	0,926	0,077
8000	0,519	0,054	0,510	0,050	0,932	0,079	0,925	0,079
9000	0,525	0,058	0,530	0,050	0,926	0,080	0,926	0,080
10000	0,520	0,059	0,530	0,060	0,924	0,080	0,926	0,080

Table 24. Average final results of the throughput parameter for the SDSC dataset

Cloudlets	GA - SDSC	PSO - SDSC	AFSA - SDSC	ECOFISH - SDSC
7395	0,057	0,056	0,057	**0,058**

3.2.8. Resource Utilization

The AFSA achieved the highest resource utilization on the uniform random dataset, while ECO-FISH performed best on the stratified random dataset (Table 25). The AFSA's resource utilization is 64.59% higher than that of the GA on the uniform dataset and 44.12% higher on the stratified dataset. ECO-FISH improves utilization by 0.01% on the stratified dataset. For the SDSC dataset (Table 26), ECO-FISH achieved the highest resource utilization, with utilization 0.87% higher than the GA's and 1.61% higher than the AFSA's.

Table 25. Average final results of the resource utilization parameter for uniform and stratified datasets

Cloudlets	GA		PSO		AFSA		ECO-FISH	
	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified
1000	50,48	37,31	22,73	25,26	85,54	34,84	87,04	34,84
2000	52,96	28,17	35,82	35,04	89,80	50,87	89,72	50,90
3000	53,83	48,87	43,52	39,14	90,86	61,22	91,39	61,20
4000	54,62	41,12	49,92	42,37	92,17	65,23	91,46	65,21
5000	55,98	49,91	54,72	43,36	92,68	67,78	92,20	67,82
6000	57,49	41,85	56,77	49,61	94,07	70,27	92,31	70,89
7000	57,42	53,97	61,13	52,05	93,54	72,14	93,50	72,21
8000	58,93	47,14	62,99	50,30	94,00	74,07	93,67	74,22
9000	59,59	50,43	65,12	49,72	94,85	75,27	94,81	75,56
10000	58,91	50,80	64,72	53,87	94,98	75,96	94,92	75,15

Table 26. Average final results of the resource utilization parameter for the SDSC dataset

Cloudlets	GA - SDSC	AFSA - SDSC	ECOFISH - SDSC
7395	35.9408	35.6785	**36.2524**

3.2.9. Imbalance Degree

ECO-FISH achieved the lowest imbalance degree on the uniform random dataset, while the AFSA performed best on the stratified random dataset (Table 27).

Table 27. Average final results of the imbalance degree parameter for uniform and stratified datasets

Cloudlets	GA		PSO		AFSA		ECO-FISH	
	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified
1000	1,755	45,084	1,612	40,113	1,475	43,553	1,494	43,552
2000	1,759	47,110	1,621	42,356	1,555	43,492	1,542	43,497
3000	1,761	45,426	1,619	42,910	1,516	43,133	1,518	43,140
4000	1,773	46,905	1,630	43,210	1,525	43,542	1,544	43,549
5000	1,771	46,178	1,632	43,163	1,651	43,443	1,652	43,442
6000	1,764	46,801	1,631	43,342	1,623	43,407	1,592	43,404
7000	1,765	46,577	1,629	43,327	1,648	43,573	1,651	43,579
8000	1,763	46,744	1,626	43,167	1,651	43,582	1,649	43,287
9000	1,760	46,526	1,628	43,343	1,632	43,553	1,635	43,585
10000	1,766	46,986	1,630	43,296	1,584	43,515	1,589	43,550

http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

The AFSA reduced the imbalance degree by 9.86% on the uniform dataset and by 6.48% on the stratified dataset compared to the GA. ECO-FISH further reduced the imbalance on the uniform dataset by 0.69% but did not improve performance on the stratified dataset. For the SDSC dataset (Table 28), ECO-FISH achieved the lowest imbalance degree. The AFSA reduced the imbalance by 21.15% compared to the GA, and ECO-FISH further reduced it by 0.21% compared to the AFSA.

Table 28. Average final results of the imbalance degree parameter for the SDSC dataset

Cloudlets	GA - SDSC	PSO - SDSC	AFSA - SDSC	ECOFISH - SDSC
7395	55,481	55,393	43,747	**43,657**

3.2.10. Total Energy Consumption

The AFSA achieved the lowest energy consumption on the uniform random dataset, while ECO-FISH performed best on the stratified random dataset (Table 29). The AFSA reduced energy consumption by 42.88% compared to the GA on the uniform dataset and by 22.75% on the stratified dataset. ECO-FISH did not improve energy consumption on the uniform dataset but reduced it by 0.01% on the stratified dataset. For the SDSC dataset (Table 30), the GA achieved the lowest energy consumption, outperforming the AFSA by 5.14% and ECO-FISH by 4.03%. ECO-FISH improved upon the AFSA's performance by reducing its energy consumption by 1.16%.

Table 29. Average final parameter total energy consumption for uniform and stratified datasets

Cloudlets	GA		PSO		AFSA		ECO-FISH	
	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified	Uniform	Stratified
1.000	10,920	89,020	11,870	132,990	6,110	80,968	6,100	80,944
2.000	20,900	216,800	20,140	209,390	11,940	141,420	12,030	141,520
3.000	31,190	246,880	28,280	291,670	17,710	201,932	17,790	201,570
4.000	41,160	364,140	36,120	371,330	23,450	258,659	23,690	258,322
5.000	50,860	397,350	44,330	432,330	29,330	317,327	29,550	317,213
6.000	60,020	532,310	52,760	498,790	35,450	377,911	35,710	378,424
7.000	70,490	538,300	60,530	576,940	41,160	438,250	41,450	437,996
8.000	80,040	653,670	68,830	659,970	47,010	498,569	47,540	498,560
9.000	88,810	713,030	76,500	733,890	52,480	556,524	52,810	557,094
10.000	99,300	785,980	85,030	804,020	58,630	633,435	59,170	633,158

Table 30. Average final parameter total energy consumption for the SDSC dataset

Cloudlets	GA - SDSC	PSO - SDSC	AFSA - SDSC	ECOFISH - SDSC
7.395	**571,050**	505,070	602,010	595,040

3.3 Discussion

The proposed ECO-FISH algorithm demonstrates several strengths across various performance metrics. For example, it minimizes the makespan and consistently matches the performance of the AFSA across all datasets (Table 31), a crucial aspect for timely task scheduling. ECO-FISH achieves the best performance in terms of average start time across all datasets, enabling tasks to commence earlier and thereby enhancing workflow efficiency. It also shows optimal performance in terms of average finish time, ensuring that operations are completed promptly. In addition, ECO-FISH effectively reduces the total scheduling length and performs comparably to the AFSA, contributing to improved resource utilization. Its performance on the simple and stratified random datasets is also comparable to the AFSA. ECO-FISH demonstrates optimal performance in maximizing resource utilization to maintain high system throughput. moreover, it achieves the best load balancing across resources, supporting system stability and preventing resource overburdening.

Table 31. Summary of algorithms with the highest performance across different datasets

Parameter	Uniform random	Stratified random	SDSC
Makespan	AFSA	ECO-FISH	ECO-FISH
Average Start Time	ECO-FISH	ECO-FISH	AFSA
Average Finish Time	ECO-FISH	ECO-FISH	AFSA
Average Execution Time	ECO-FISH	GA	AFSA
Total Wait Time	AFSA	ECO-FISH	AFSA
Total Scheduling Length	ECO-FISH	ECO-FISH	ECO-FISH
Throughput (max.)	ECO-FISH	ECO-FISH	AFSA
Resource Utilization (max.)	AFSA	ECO-FISH	ECO-FISH
Total Energy Consumption	AFSA	ECO-FISH	PSO
Imbalance Degree	ECO-FISH	AFSA	ECO-FISH

Despite its numerous strengths, the proposed ECO-FISH algorithm also has areas that warrant further improvement. For example, in terms of average execution time, while ECO-FISH performs

http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

competitively on the uniform random dataset, it is outperformed by the GA on the stratified random dataset and by the AFSA on the SDSC dataset. The total wait time produced by ECO-FISH shows the best performance on the uniform random and SDSC datasets. However, the AFSA performs better on the stratified random dataset. Furthermore, while ECO-FISH performs well in reducing total energy consumption on the uniform and stratified random datasets, it is surpassed by PSO on the SDSC dataset.

The OBL component within ECO-FISH was not designed or evaluated as a standalone optimization method. Instead, it functions to enhance AFSA's global exploration capability by introducing opposite task–VM mappings that help the swarm escape local optima and maintain diversity in the search space. OBL alone lacks the iterative swarm behaviors—prey, follow, and swarm—that enable AFSA to perform adaptive scheduling and convergence-driven refinement. Therefore, comparing OBL as an independent baseline would not yield meaningful results. Instead, the evaluation focuses on demonstrating how integrating OBL with AFSA improves convergence stability and optimization performance across various workload distributions. The results confirm that the synergy between AFSA's local refinement and OBL's global diversification produces more balanced, robust, and efficient scheduling in dynamic cloud environments.

4. Conclusions

We developed a scheduling algorithm for cloud computing environments to improve resource management optimization by combining AFSA with OBL. The proposed ECO-FISH model leverages AFSA's swarm intelligence for efficient local search and OBL's capability for global diversification. Experimental results show that AFSA alone outperforms the GA, reducing makespan by 28–45%, increasing throughput by 34–84.9%, and improving resource utilization by 44.12–64.59%. Integrating OBL further enhances these metrics by up to 1.6% in utilization and 1.5% in makespan, demonstrating its contribution to avoiding premature convergence and achieving more stable scheduling outcomes. On the SDSC dataset, ECO-FISH achieves a 12.84% lower makespan, 27.54% higher utilization, and a 25.03% improvement in performance. These improvements are statistically significant and yield substantial operational benefits in large-scale cloud environments. Overall, AFSA performs best on uniform workloads, while the combined AFSA-OBL model is more effective for heterogeneous or highly variable datasets, demonstrating greater robustness, balanced load distribution, and adaptability in dynamic cloud resource management.

Author Contributions

A. M. Shiddiqi: Conceptualization, formal analysis, funding acquisition, supervision, writing – original draft. H. T. Ciptaningtyas: Conceptualization, formal analysis, investigation, methodology, validation, and writing – review & editing. J. Leonardo: Data curation, investigation, methodology, and software. F. Rahma: Project administration, validation, and writing – review & editing.

Acknowledgment

This research was supported by the Departmental Research Grant Batch 2 from Institut Teknologi Sepuluh Nopember (ITS), Indonesia year 2024 under Grant Number 2004/PKS/ITS/2024.

Declaration of Competing Interest

We declare that we have no conflict of interest.

References

[1] K. Ma, Z. Cai, X. Yan, Y. Zhang, Z. Liu, Y. Feng, and C. Li, "PPS: Fair and efficient black-box scheduling for multi-tenant GPU clusters," Parallel Comput., vol. 120, p. 103082, Jun. 2024, doi: https://doi.org/10.1016/j.parco.2024.103082.

[2] H. T. Ciptaningtyas, A. M. Shiddiqi, D. Purwitasari, F. D. Rosyadi, and M. N. Fauzan, "Multi-objective Task Scheduling Algorithm in Cloud Computing Using Improved Squirrel Search Algorithm," *Int. J. Intell. Eng. Syst.*, vol. 17, no. 1, pp. 895–912, 2024, doi: https://doi.org/10.22266/ijies2024.0229.74.

[3] I.Z. Yakubu, L. Muhammed , Z. A. Musa, Z. I. Matinja, and I. M. Adamu, "A Multi Agent Based Dynamic Resource Allocation in Fog-Cloud Computing Environment," *Trends in Sciences.*, vol. 18 (22):, 413, 2021, doi: https://doi.org/10.48048/tis.2021.413.

[4] Z. Zhou, F. Li, H. Zhu, H. Xie, J. H. Abawajy, and M. U. Chowdhury, "An improved genetic

 http://doi.org/10.26594/register.v11i2.5340

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

algorithm using greedy strategy toward task scheduling optimization in cloud environments," *Neural Comput. Appl.*, vol. 32, no. 6, pp. 1531–1541, 2020, doi: 10.1007/s00521-019-04119-7.

[5] L. Abualigah and M. Alkhrabsheh, "Amended hybrid multi-verse optimizer with genetic algorithm for solving task scheduling problem in cloud computing," *J. Supercomput.*, vol. 78, no. 1, pp. 740–765, 2022, doi: 10.1007/s11227-021-03915-0.

[6] Z.-J. Wang, J. Yang, Y. Liu, Z. Guo, Y. Zhang, and H. Zhao, "Dynamic Group Learning Distributed Particle Swarm Optimization for Large-Scale Optimization and Its Application in Cloud Workflow Scheduling," IEEE Trans. Cybern., vol. 50, no. 6, pp. 2715–2729, 2020, doi: https://doi.org/10.1109/TCYB. 2019.2933499.

[7] Y. Dasril, M. A. Muslim, M. F. A. Hakim, J. Jumanto, and B. Prasetiyo, "Credit Risk Assessment in P2P Lending Using LightGBM and Particle Swarm Optimization," *J. Ilm. Teknol. Sist. Inf.*, vol. 9, no. 1, pp. 18–28, 2023.

[8] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li, "A scheduling scheme in the cloud computing environment using deep Q-learning," *Inf. Sci. (Ny).*, vol. 512, pp. 1170–1191, 2020, doi: https://doi.org/10.1016/j.ins.2019.10.035.

[9] S. G. Domanal, R. M. R. Guddeti, and R. Buyya, "A Hybrid Bio-Inspired Algorithm for Scheduling and Resource Management in Cloud Environment," *IEEE Trans. Serv. Comput.*, vol. 13, no. 1, pp. 3–15, 2020, doi: 10.1109/TSC.2017.2679738.

[10] A. P. Kurniawan, M. N. Ashar, F. Hidayat, S. Salsabila, P. T. W. Gautama, A. M. Shiddiqi, and H. Studiawan, "Performance Evaluation for Deploying Dockerized Web Application on AWS, GCP, and Azure," in *2023 IEEE Int. Conf. Control, Electronics and Computer Technology (ICCECT)*, pp. 346–350, 2023, doi: https://10.1109/ICCECT57938.2023.10140775.

[11] S. Pandiyan, T. S. Lawrence, S. V., M. Ramasamy, Q. Xia, and Y. Guo, "A performance-aware dynamic scheduling algorithm for cloud-based IoT applications," *Comput. Commun.*, vol. 160, pp. 512–520, 2020, doi: https://doi.org/10.1016/j.comcom.2020.06.016.

[12] A. S. S. Ansyah, M. Arifin, M. B. Alfan, M. V. Suriawan, N. H. Farhansyah, A. M. Shiddiqi, and H. Studiawan, "MQTT Broker Performance Comparison between AWS, Microsoft Azure and Google Cloud Platform," in 2023 International Conference on Recent Trends in Electronics and Communication (ICRTEC), 2023, pp. 1–6, doi: https://doi.org/10.1109/ICRTEC56977.2023.10111870.

[13] N. Arora and R. K. Banyal, "Hybrid scheduling algorithms in cloud computing: a review," *Int. J. Electr. Comput. Eng. (IJECE); Vol 12, No 1 Febr. 2022*, 2022, doi: 10.11591/ijece.v12i1.pp880-895.

[14] H. Hendy, M. I. Irawan, I. Mukhlash, and S. Setumin, "A Bibliometric Analysis of Metaheuristic Research and Its Applications," *J. Ilm. Teknol. Sist. Inf.*, vol. 9, no. 1, pp. 1–17, 2023.

[15] M. S. A. Khan and R. Santhosh, "Task scheduling in cloud computing using hybrid optimization algorithm," *Soft Comput.*, vol. 26, no. 23, pp. 13069–13079, 2022, doi: 10.1007/s00500-021-06488-5.

[16] H. Liu, "Research on cloud computing adaptive task scheduling based on ant colony algorithm," *Optik (Stuttg).*, vol. 258, p. 168677, 2022, doi: https://doi.org/10.1016/j.ijleo.2022.168677.

[17] M. Neshat, G. Sepidnam, M. Sargolzaei, and A. N. Toosi, "Artificial fish swarm algorithm: a survey of the state-of-the-art, hybridization, combinatorial and indicative applications," *Artif. Intell. Rev.*, vol. 42, no. 4, pp. 965–997, 2014, doi: 10.1007/s10462-012-9342-2.

[18] A. Nekooei-Joghdani and F. Safi-Esfahani, "Dynamic scheduling of independent tasks in cloud computing applying a new hybrid metaheuristic algorithm including Gabor filter, opposition-based learning, multi-verse optimizer, and multi-tracker optimization algorithms," *J. Supercomput.*, vol. 78, no. 1, pp. 1182–1243, 2022, doi: 10.1007/s11227-021-03814-4.

[19] S. K. Mishra, B. Sahoo, and P. P. Parida, "Load balancing in cloud computing: A big picture," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 32, no. 2, pp. 149–158, 2020, doi: https://doi.org/10.1016/j.jksuci.2018.01.003.

[20] P. Barredo and J. Puente, "Precise makespan optimization via hybrid genetic algorithm for scientific workflow scheduling problem," *Nat. Comput.*, vol. 22, no. 4, pp. 615–630, 2023, doi: 10.1007/s11047-023-09950-5.

[21] M. Ghobaei-Arani, A. Souri, and A. A. Rahmanian, "Resource Management Approaches in Fog Computing: a Comprehensive Review," *J. Grid Comput.*, vol. 18, no. 1, pp. 1–42, 2020, doi:

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
11 (2) 2025 106-122

10.1007/s10723-019-09491-1.

[22] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet of Things*, vol. 12, p. 100273, 2020, doi: https://doi.org/10.1016/j.iot.2020.100273.

[23] X. Ye, J. Li, S. Liu, J. Liang, and Y. Jin, "A hybrid instance-intensive workflow scheduling method in private cloud environment," *Nat. Comput.*, vol. 18, no. 4, pp. 735–746, 2019, doi: 10.1007/s11047-016-9600-3.

[24] X. Tang, Y. Liu, J. Zhang, Y. Chen, and H. Zhang, "Cost-Efficient Workflow Scheduling Algorithm for Applications With Deadline Constraint on Heterogeneous Clouds," IEEE Trans. Parallel Distrib. Syst., vol. 33, no. 9, pp. 2079–2092, 2022, doi: https://doi.org/10.1109/TPDS.2021.3134247.

[25] T. Li, F. Yang, D. Zhang, and L. Zhai, "Computation Scheduling of Multi-Access Edge Networks Based on the Artificial Fish Swarm Algorithm," *IEEE Access*, vol. 9, pp. 74674–74683, 2021, doi: 10.1109/ACCESS.2021.3078539.

[26] P. S. Rawat, P. Dimri, P. Gupta, and G. P. Saroha, "Resource provisioning in scalable cloud using bio-inspired artificial neural network model," *Appl. Soft Comput.*, vol. 99, p. 106876, 2021, doi: https://doi.org/10.1016/j.asoc.2020.106876.

[27] S. Mahdavi, S. Rahnamayan, and K. Deb, "Opposition based learning: A literature review," *Swarm Evol. Comput.*, vol. 39, pp. 1–23, 2018, doi: https://doi.org/10.1016/j.swevo.2017.09.010.

[28] Y. Zhu and H. Gao, "Improved Binary Artificial Fish Swarm Algorithm and Fast Constraint Processing for Large Scale Unit Commitment," *IEEE Access*, vol. 8, pp. 152081–152092, 2020, doi: 10.1109/ACCESS.2020.3015585.

[29] F. Pourpanah, R. Wang, C. P. Lim, X.-Z. Wang, and D. Yazdani, "A review of artificial fish swarm algorithms: recent advances and applications," *Artif. Intell. Rev.*, vol. 56, no. 3, pp. 1867–1903, 2023, doi: 10.1007/s10462-022-10214-4.

[30] L. Wang, L. An, J. Pi, M. Fei, and P. M. Pardalos, "A diverse human learning optimization algorithm," *J. Glob. Optim.*, vol. 67, no. 1, pp. 283–323, 2017, doi: 10.1007/s10898-016-0444-2.

[31] H. Krishnaveni and V. S. Janita, "Modified Artificial Fish Swarm Algorithm for Efficient Task Scheduling in Cloud Environment," *Int. J. Comput. Sci. Eng.*, vol. 7, no. 5, pp. 1363–1371, 2019, doi: 10.26438/ijcse/v7i5.13631371.

[32] C. Li, J. Tang, T. Ma, X. Yang, and Y. Luo, "Load balance based workflow job scheduling algorithm in distributed cloud," *J. Netw. Comput. Appl.*, vol. 152, p. 102518, 2020, doi: https://doi.org/10.1016/j.jnca.2019.102518.

[33] H. Krishnaveni, V. S. Janita, T. Earheart, and N. Wilkins-Diehr, "SDSC Blue Horizon," 2000. https://www.cs.huji.ac.il/labs/parallel/workload/l sdsc blue/index.html (accessed Jul. 20, 2024).

 http://doi.org/10.26594/register.v11i2.5340