ISSN: 2182-2069 / E-ISSN: 2182-2077

An Energy-aware Dynamic Scheduling Algorithm for Optimizing Workflows Under Budget-constraints

C.K. Sripavithra[1*], and Dr.V.B. Kirubanand[2]

[1*]Central Campus, Christ (Deemed to be University), Bangalore, India; Maharani's Science College for Women, Mysore, India. sripavithra.ck@res.christuniversity.in, https://orcid.org/0000-0003-4325-4564

[2]Central Campus, Christ (Deemed to be University), Bangalore, India. kirubanand.vb@christuniversity.in, https://orcid.org/0000-0003-0792-548X

Received: November 26, 2024; Revised: January 07, 2025; Accepted: January 25, 2025; Published: February 28, 2025

Abstract

The provision of cloud computing offers an untapping scalability and elasticity which is best suited for the execution of user tasks and complicated scientific workflows. Regardless, the big problem of workflow scheduling under a user-specified budget still prevails as a result of the task inter-dependencies and the resource diversity. This research proposes a hybrid Energy-Aware Enhanced Salp Swarm Algorithm (EA-ESSA), designed to dynamically schedule tasks while adhering to user-specified budget constraints. This technique supports dynamic scheduling using task duplication in idle time spots and integrates APIs for real-time spot pricing. This proposed technique also minimizes makespan and energy consumption by improving resource utilization. The algorithm's performance was exhaustively experimented with using both simulated workloads and actual HPC2N datasets. The simulation results show significant advancements in the makespan, resource utilization and energy consumption compared to existing algorithms like ACO, GA, PSO, and MOTSWAO. This research benefits cloud environments comprising complex, unpredictable workflows by cutting environmental effects and shrinking processing expenses.

Keywords: Scientific Workflows, Task Duplication, Idle Spots, Budget Constraint, Makespan, Resource Utilization, Energy Consumption.

1 Introduction

Cloud computing has thoroughly transformed how computations and procedures are processed online. By employing a pay-as-you-go feature, this paradigm eradicates the necessity of infrastructure for enterprises and researchers. It is especially suitable for experimental processing of complex scientific workflows that might be data or compute-intensive functions demanding considerable processing power (Albtoush et al., 2024).

Banking on the upper hand of cloud computing, various big data applications from physics, genomics, and bioinformatics are processed in the cloud (Saravanan et al., 2022) (Rani & Tukkoji, 2024). These are constituted by high computations and inter-dependencies and are depicted as Directed Acyclic Graphs (DAG). These workflows require appropriate scheduling and resource management to

Journal of Internet Services and Information Security (JISIS), volume: 15, number: 1 (February), pp. 182-199. DOI: *10.58346/JISIS.2025.I1.012*

*Corresponding author: Central Campus, Christ (Deemed to be University), Bangalore, India; Maharani's Science College for Women, Mysore, India.

meet the optimal performance and economic conditions (Ulabedin et al., 2024). Practical scheduling methods are vital for lowering the completion time and utilizing less energy while handling vast data (Stevovic et al., 2023; Ghafir et al., 2024).

Task scheduling algorithms must consider the dependencies among the tasks and restrict the data movement among them to reduce cost and latency (Sandhu et al., 2023). Appropriate allocation of resources based on the funds allocated is crucial. Efficient scheduling plans aim to minimize expenditures by allocating resources as effectively as feasible while sticking to client-given financial restrictions (Wu et al., 2022). Task replication reduces performance latencies and improves reliability by leveraging underutilization to duplicate critical actions in vacant spaces (Yao et al., 2021).

2 Related Works

One of the foremost crises of executing workflows on cloud computing is the budget restriction and the energy consumption of devices in the data center (Canon et al., 2020; Giliberto et al., 2019; Chen et al., 2017). Currently, much research is going on in this direction (Hilman et al., 2020; Wang & Shi, 2014; Cao et al., 2019). The primary purpose of this study is to develop approaches to address the rising complexity of diverse resources and dynamic executions in cloud systems.

For instance, to balance performance, expense, and energy use, the heterogeneous earliest completion time (HEFT) strategy offered an expense decrease of up to 23% over other approaches (Hua et al., 2022).

In mixed trends, energy-reducing methods such as the Modified Firefly Optimization Algorithm (ModFOA) demonstrate improved asset use and decreased make span. For effectual scheduling, this plan emphasized the importance of comprising edge computing resources (Alsadie & Alsulami, 2024; Yao et al., 2021). To reduce inactive resource periods and adhere to financial restrictions, task replication-based strategies were also used for selective task duplication, improving resource usage by 31.6% and shortening workflow makespan by up to 17.4% (Giliberto et al., 2019).

Advanced heuristic and metaheuristic techniques have been investigated, including ant colony optimization, particle swarm optimization, and genetic algorithms (ChithraDevi et al., 2024) (Prasath, 2024). These techniques tackle multi-objective cloud scheduling issues like cost, performance, and energy efficiency trade-offs. For instance, the ACO-LB algorithm efficiently balanced load and reduced scheduling makespan (Xue et al., 2014).

The Duplication and Insertion-based List Scheduling (DILS) technique dynamically schedules tasks based on job copying, job insertion, and completion time estimation to reduce the makespan. This method expedites work completion and reduces channeling times by efficiently utilizing idle spots (Fan et al., 2020). A similar approach centered on asset variety and task interdependence to enhance resource utilization in multi-job task scheduling (Shi et al., 2021).

For heterogeneous cloud environments, a task duplication-based scheduling strategy optimized resource utilization and reduced workflow execution time. This method intentionally duplicates tasks and determines task priority (Gupta et al., 2016). To increase the stability and caliber of task scheduling in multi-task scenarios, an adaptive PSO-based algorithm combined disturbance and disorder operators (Zhang et al., 2018).

State-of-the-art techniques that balanced processing duration and expenditure, such as the Modified Cuckoo Search Algorithm, exhibited substantial gains over traditional heuristic algorithms (Shi et al., 2021).

Concerning performance standards like makespan of 97% and asset utilization of 99%, the Levy Flight Osprey Optimization Algorithm (LFOOA) functions more promising than other methods like Particle Swarm Optimization and Grey Wolf Optimization (Chintale et al., 2024). Sustainable operations rely on work scheduling plans for green cloud computing that underline energy-efficient algorithms (Sadotra et al., 2025). The strategies of AHP and Multiple-Criteria Decision-Making (MCDM) algorithms depict lowering delay periods and enhancing resource utilization (Goyal et al., 2024; El-Saadawi et al., 2024).

Similarly, execution time and resource spots are contemplated by a Task Scheduling Model (TSM) that settles resource fragmentation problems (Kamalam et al., 2023). Merging the Krill Herd technique with the Estimate of Distribution Algorithm (EDA-KrillHerd) also indicated superior performance in handling large-scale schemes (Muniyappa & Hattibelagal, 2023).

This review collectively shows the importance of using task duplication, adaptable algorithms, and mixed computational methods to maximize plans in cloud computing environments (Zigui et al., 2024).

3 Problem Statement

The extensive literature study shows that few studies have effectively managed the exploitation of inactive resource slots due to the dependence on workflow tasks. Even with a comprehensive survey of workflow scheduling, significant improvement needs to be addressed in resource usage while remaining within financial constraints. Existing algorithms skip the crucial energy efficiency concerns and economic constraints, creating a substantial gap in achieving cost-effective solutions (Tan et al., 2024).

To address the twin concerns of environmental and economic sustainability, this project aims to create and implement a workable algorithm that schedules activities in cloud environments as effectively as feasible while adhering to budgetary constraints.

4 Material and Methods

Problem Formulation

The main objectives of this research on scheduling of workflows include

a. Design a Modified Task Duplication Scheduling Algorithm (M-TDSA) by using Task Duplication Scheduling Algorithm (TDSA) (Zhang et al., 2020) by strengthening with duplication control mechanism and involving heterogeneous resources.

b. Develop a Multi objective Salp Swarm Algorithm (M-SSA) by using Salp Swarm Algorithm SSA (Mirjalili et al., 2017) by including multi-objective criteria with expense constraint.

c. Combine M-TDSA and M-SSA, to design an Enhanced Salp Swarm Algorithm (ESSA) to reduce makespan and improve resource utilization considering different number of tasks, number of VM, CPU, memory, resource cost, number of resources, and types of resources under budget constraints.

d. Integrate an energy aware module in ESSA to propose an Energy Aware - Enhanced Salp Swarm Algorithm (EA-ESSA), an energy-efficient workflow scheduling algorithm under client-specified budget constraints and also use APIs for real-time spot pricing to further optimize resource allocation and reduce expenses (Surendar, 2024).

e. Experiment EA-ESSA with synthetic task distributions and scientific HPC2N worklogs. Simulate under user-specified budget limits to assess the algorithm's performance.

Cloud Scheduling Model

Figure 1 represents a Cloud Scheduling Model that illustrates the flow of processes and components involved in scheduling workflows in cloud environments.

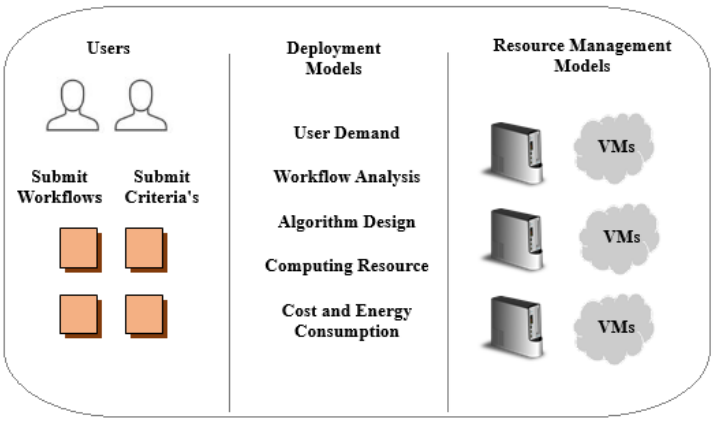

Figure 1: Cloud Scheduling Model

1. Users

Submit Workflows: Users provide workflows consisting of tasks that need to be executed. A workflow is typically modeled as a Directed Acyclic Graph (DAG) with tasks (nodes) and their dependencies (edges).

Submit Criteria: Users specify requirements or criteria for the workflow, in this case:

- Budget for resource usage.
- Quality of Service (QoS) metrics like makespan, resource utilization, load balancing and energy consumption or throughput.

2. Deployment Models

This section represents the core analysis and decision-making processes for scheduling workflows:

User Demand: Refers to the requirements or preferences submitted by the users, in this case it will be the expense constraint.

Workflow Analysis: The workflows are analyzed to identify:

- Task dependencies.
- Computational and communication costs.

Algorithm Design: Refers to the dynamic scheduling algorithm employed to map tasks onto available resources (e.g., VMs) efficiently.

Computing Resource: The available resources (e.g., virtual machines or servers) are analyzed to determine suitability for executing tasks based on factors like processing power, cost or type.

Cost and Energy Consumption: The deployment model evaluates:

Cost: How much it will cost to execute the workflows based on the cloud provider's pricing model.

Energy Consumption: Focuses on energy-efficient scheduling to minimize environmental impact and operational expenses.

Real-Time Pricing Integration: Current cost information for cloud resources is gathered as part of the deployment model using APIs for real-time spot pricing. This data is utilized to continually choose the most cost-effective resources and adhere to client-given budgetary constraints during the process study and design phases.

5 Proposed Methodology

Steps

a. Initial Scheduling (M-TDSA)

M-TDSA first manages task placement by concentrating on reducing the time-dependent tasks to communicate. Tasks are replicated when needed to save expense on communication between various cloud assets.

b. Optimization Phase (ESSA)

ESSA is used to arrange tasks on available cloud resources better when the M-TDSA generates a schedule, considering the balance of work and performance.

c. Energy Aware Model

The energy model added in ESSA (EA-ESSA) explores different solutions to improve resource utilization and ensure the system operates efficiently under varying cloud workloads with reduced energy consumption.

d. Incorporation of Real-Time Spot Pricing APIs

The EA-ESSA algorithm incorporates APIs for real-time spot pricing to further optimize resource allocation and reduce expenses.

Task Duplication-Based Scheduling Algorithm (TDSA) (Yao et al., 2021):

The basic Task Duplication-Based Scheduling Algorithm (TDSA) is designed to optimize workflow scheduling on cloud platforms by minimizing makespan while ensuring budget constraints. It leverages redundancy to reduce inter-task communication overhead, minimize idle time on processors, and enhance overall parallel execution.

TDSA Algorithm

Algorithm TDSA (DAG G, Resources R)

Input:

G = (V, E) // A DAG with tasks V and edges E representing dependencies
R = {r1, r2, ..., rn} // Set of available resources (processors)
ExecutionTime(v, r) // Execution time of task v on resource r
CommunicationCost(v1, v2) // Communication cost between tasks v1 and v2

Output:

```
Schedule S  // A mapping of tasks to resources and their start times

Step 1: Initialization
  - Initialize ready_task_set ← {tasks with no predecessors}
  - Initialize S ← empty schedule
  - Initialize available_time[r] ← 0 for each resource r in R

Step 2: Sub-budget allocation (if applicable)
  - Distribute sub-budgets for all unscheduled tasks (optional, based on budget constraints)

Step 3: Scheduling loop
  While ready_task_set is not empty do:
    For each task t in ready_task_set:
      For each resource r in R:
        - Calculate the earliest start time of t on r considering both:
          a) Data transfer time (if t's predecessors are on different resources)
          b) Duplication of predecessors on the same resource to avoid communication cost
        - If duplication reduces start time, duplicate predecessor tasks on r
        - Calculate the finish time of task t on resource r
      End for

      - Select the resource r_min where t finishes the earliest
      - Schedule t on r_min with its start time and finish time
      - Update available_time[r_min] with t's finish time

    End for

    - Update ready_task_set by adding tasks whose predecessors are now scheduled
  End while

Step 4: Output final schedule S

End Algorithm
```

Modified Task Duplication-Based Scheduling Algorithm (M-TDSA):

In the basic TDSA, a few enhancements are done to

- **Reduce Resource Overhead Due to Duplication**
  - Introduce a dynamic task duplication control mechanism where duplication is only allowed when it significantly reduces the communication delay.
  - Here an intelligent trade-off between duplication and communication cost is considered, limiting unnecessary duplications.
- **Including Resource Heterogeneity**
  - Extend TDSA to consider heterogeneous resources by factoring in the execution speed differences across processors.

- Incorporate resource profiling so that task-resource mapping is optimized based on both task type and processor speed.

The enhancements are presented in the subroutines: DynamicTaskDupControl() and Hetero() and the Figure 2 shows the flow of processing of M-TDSA.

```
Subroutine DynamicTaskDupControl (Task t, Resource r, Predecessors P, DuplicationThreshold δ)
  Input:
    t: Current task to be scheduled
    r: Resource where the task is being considered
    P: Set of predecessor tasks of t
    DuplicationThreshold δ: Minimum reduction in communication delay to allow duplication

  Output:
    ResourceAssignment  // Resource where task t will be scheduled

  Step 1: Calculate the standard communication cost
    For each predecessor p in P:
      CommCost[p] = CalculateCommunicationCost(p, t, r)  // Cost if predecessor p is on a different resource
  Step 2: Check for duplication viability
    For each predecessor p in P:
      if (p is not already scheduled on r) then:
        DuplicationCost[p] = ExecutionTime(p, r)  // Cost of duplicating p on r
       // Calculate benefit of duplicating p in terms of reduced communication delay
        Benefit[p] = CommCost[p] - DuplicationCost[p]
        // Allow duplication only if the benefit exceeds the predefined threshold δ
        if Benefit[p] > δ then:
          DuplicateTask(p, r)  // Duplicate the predecessor p on resource r
        else:
          Continue without duplicating p
      else:
        // If p is already on r, no need to duplicate
        Continue
  Step 3: Schedule task t on resource r
    ScheduleTask(t, r)
  Return ResourceAssignment
End Subroutine
```

```
Subroutine Hetero (DAG G, Resources R)
  Input:
    G = (V, E)  // A DAG with tasks V and edges E representing dependencies
    R = {r1, r2, ..., rn}  // Set of available resources (processors) with different speeds
    ExecutionTime(v, r)  // Execution time of task v on resource r, based on r's speed
    CommunicationCost(v1, v2)  // Communication cost between tasks v1 and v2

  Output:
```

```
    Schedule S  // A mapping of tasks to resources and their start times

  Step 1: Initialization
    - Initialize ready_task_set ← {tasks with no predecessors}
    - Initialize S ← empty schedule
    - Initialize available_time[r] ← 0 for each resource r in R
    - Initialize speed[r] ← performance factor of resource r  // Higher speed means faster execution

Step 2: Scheduling loop
    While ready_task_set is not empty do:
      For each task t in ready_task_set:
        For each resource r in R:
          // Calculate the earliest start time on resource r considering resource speed
          EarliestStart[r] = max(available_time[r], FinishTimes of predecessors on other resources)
          // Calculate adjusted execution time based on resource r's speed
          AdjustedExecTime = ExecutionTime(t, r) / speed[r]  // Faster resources have shorter exec time

          // Calculate finish time of task t on resource r
          FinishTime[r] = EarliestStart[r] + AdjustedExecTime

          // If predecessors are on different resources, consider communication cost
          For each predecessor p of t:
            if (p is scheduled on a different resource than r) then:
              FinishTime[r] += CommunicationCost(p, t)  // Add communication delay
          End for
        // Select the resource r_min where the adjusted finish time is the earliest
        r_min = argmin(FinishTime[r] for all r in R)

        // Schedule task t on r_min with its start time and finish time
        ScheduleTask(t, r_min, EarliestStart[r_min], FinishTime[r_min])
        available_time[r_min] = FinishTime[r_min]  // Update resource availability after scheduling

      End for

      // Update ready_task_set by adding tasks whose predecessors are now scheduled
      UpdateReadyTasks(ready_task_set)
    End while

  Step 3: Output final schedule S
End Subroutine
```

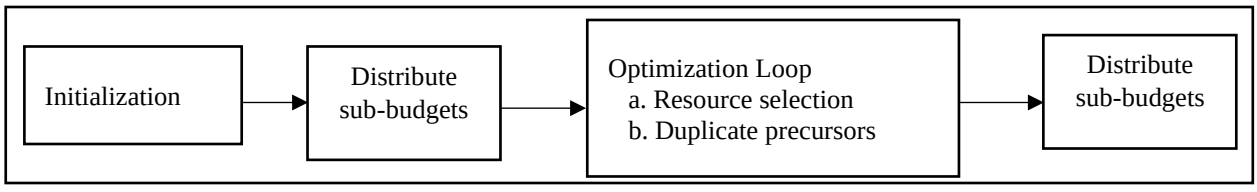

Figure 2: Flow Diagram of M-TDSA

Salp Swarm Algorithm (SSA) (Mirjalili et al., 2017)

This algorithm draws inspiration from the swarming action of salps, which are jellyfish-like sea invertebrates. To effectively navigate and find food in their habitat, salps have a unique behavior that allows them to construct chains while traveling in the ocean.

This collective behavior, in which individual salps cooperate to explore and take advantage of the search space, is replicated by the algorithm. The salp distribution is divided into two groups. The leader is the first salp in the food chain; the rest are followers. SSA generates a collection of solutions (X) of dimension (D).

Multiobjective Salp Swarm Algorithm (M-SSA)

Basic SSA struggles to balance multiple conflicting objectives effectively. Multi-objective optimization requires algorithms to handle trade-offs between competing objectives, and SSA's single-objective structure requires adaptation to work in multi-objective problems.

Algorithm Multiobjective Salp Swarm Algorithm (M-SSA)

1. Initialize the salp population P with size N.
2. Define the number of iterations Max_Iter and the objectives f1(x), f2(x), ..., fm(x).
3. Randomly initialize the position of each salp within the search space.

4. FOR each iteration t = 1 to Max_Iter DO
   5. FOR each salp i in population P DO
      6. Evaluate the fitness of each salp for all objectives f1(x), f2(x), ..., fm(x).
      7. Perform Non-Dominated Sorting to classify salps based on Pareto dominance.
      8. Calculate DI to preserve diversity in Pareto-optimal solutions.
      9. IF i == 1 (Leader salp) THEN
         10. Update the leader's position using **Sine-Cosine Movement**:
            - r1 = random value in [0, 2π]
            - r2 = random value in [0, 2]
            - IF r2 < 0.5 THEN
               - leader_position = current_position + r1 * sin(r1) * (upper_bound - lower_bound)
            - ELSE
               - leader_position = current_position + r1 * cos(r1) * (upper_bound - lower_bound)
         11. Adjust the leader's movement based on non-dominated sorting and objective preferences to explore different Pareto front regions.
12. ELSE (Follower salps)
         13. Update the follower salps' position using SSA's leader-following mechanism:
            - follower_position = 0.5 * (position_i + position_(i-1))
      14. Ensure all salps stay within the search space boundaries.

   15. Update the Pareto front based on non-dominated solutions.
   16. Apply crowding distance for solution diversity maintenance.

17. END FOR

18. Return the final Pareto front representing the best trade-offs between objectives.
End Algorithm

Enhanced Salp Swarm Algorithm (ESSA)

The hybrid approach of combining M-TDSA and M-SSA results in the Enhanced Salp Swarm Algorithm (ESSA) algorithm which dynamically schedules and shortens workflows execution within the budget and also maximizes resource usage by considering the heterogeneous resources.

Energy Aware – Enhanced Salp Swarm Algorithm (EA-ESSA):

An energy aware model is integrated to the E-SSA for reducing the energy consumption. Considering the energy utilization pattern presented in (Fan et al., 2007) for a PM, the energy utilization of the M_j, denoted as $Po_j(.)$, is given with

$$Po_j(t) = P_j^{idle} + (P_j^{max} - P_j^{idle})C_j(t) \quad (1)$$

The processing consumption of the physical machine for period p, given as $C_j(t)$, is meant as the mean fraction of total assigned processing powers of $n_j(g)$. VMs is assigned to the M_j. Using APIs, the algorithm dynamically seeks the most recent cloud resource pricing, guaranteeing economical job scheduling and resource consumption within financial limitations. Adapting to changes in spot instance prices enables the scheduler to optimize performance while lowering costs.

Algorithm Modified Salp Swarm Algorithm + Energy Aware Model

1. Initialize parameters:
   - Initialize the number of salps (N)
   - Initialize the position of each salp (task allocation on cloud resources)
   - Set maximum iterations (Max_iter)
   - Initialize energy consumption model parameters (e.g., energy per resource, energy per task)
   - Define cloud resources with their energy consumption profiles (CPU, memory, bandwidth, etc.)

2. Define objective function:
   - Objective = Minimize (Energy consumption + Makespan)
   - Energy consumption = Sum of energy consumed by all cloud resources during task execution

3. Initialize the leader (best salp) and followers (other salps):
   - Evaluate each salp's position (task allocation) based on the objective function (considering both makespan and energy consumption).
   - Set the best salp (leader) as the solution with the minimum objective value.
4. While (iter < Max_iter):
   - Update the position of the leader salp using the formula:
     X_leader = X_leader + c1 * (Food_source - X_leader)
     Where:
     - X_leader: Position of the leader (task-resource allocation).
     - c1: Convergence factor that decreases over iterations.
     - Food_source: Optimal task-resource allocation based on the objective function.
   - For each follower salp:
     - Update position using the following formula:
       X_follower_i = (X_follower_i + X_follower_(i-1)) / 2
       - Evaluate the new position based on the objective function (minimizing energy and makespan).
   - Calculate energy consumption for each salp (resource allocation):
     Energy = ∑ (Power_usage(Resource_j) * Task_duration(Task_i on Resource_j))
     Where:

- Power_usage(Resource_j): Power consumption rate of resource j.
- Task_duration(Task_i on Resource_j): Time to execute task i on resource j.
- Calculate makespan for the current solution:
  Makespan = max(Finish_time_of_all_tasks)
- Evaluate the total fitness of each salp:
  Fitness = w1 * Energy + w2 * Makespan
  Where:
- w1, w2: Weights to balance energy consumption and makespan.
- If a follower salp's position is better than the leader:
- Update the leader with the best salp.

5. Update convergence factor (c1) to slowly reduce exploration and focus on exploitation.

6. Repeat until maximum iterations (Max_iter) or convergence criterion is met.

7. Output the optimal solution:
- Optimal task allocation minimizing both energy consumption and makespan.
- Report the total energy consumed and makespan for the solution.

End Algorithm

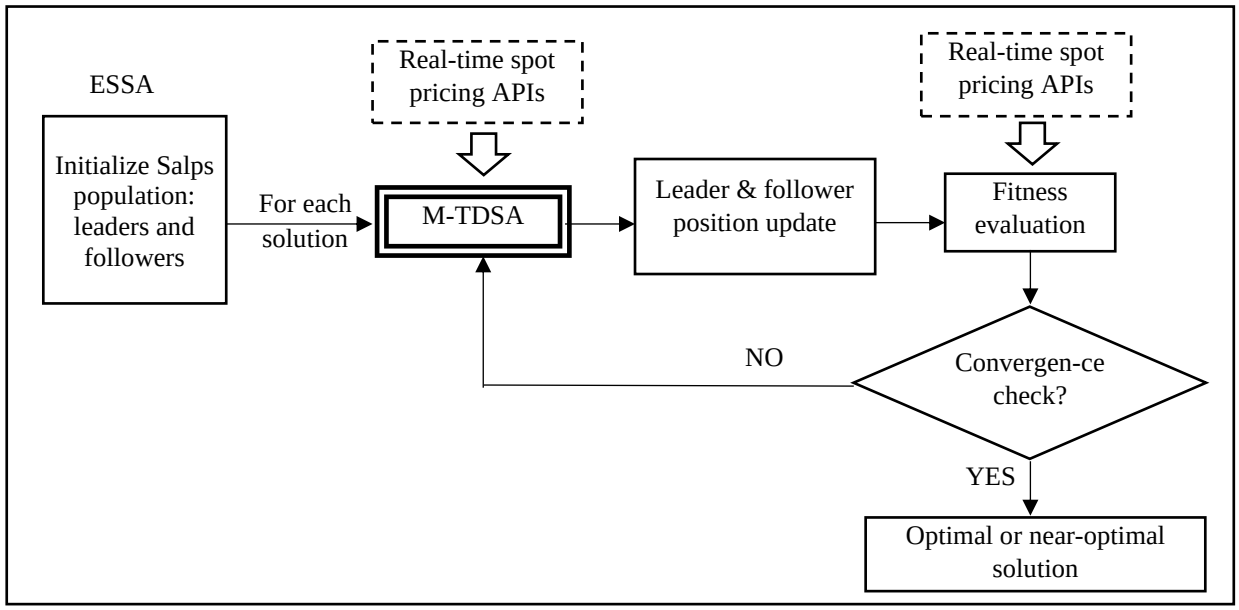

Figure 3: Flow Diagram of EA-ESSA

In figure 3 shows the flow diagram of EA-ESSA above.

6 Result and Analysis

In this section, we have elaborated on the simulations performed using CloudSim, a popular Java-based open-source simulation framework developed for modeling and simulating cloud computing infrastructures and services (Calheiros et al., 2011). We have considered workload using synthetic datasets with varied distributions and also by using real worklogs of HPC2N (Goyal et al., 2024). The proposed EA-ESSA is compared with other baseline approaches i.e. ACO, GA, PSO and MOTSWAO algorithms with respect to asset usage as well as the makespan under budget constraints.

Simulation Layout Arrangement

12 GB of RAM, an i5 processor, and a Windows operating system environment that enables virtualization make up the actual host setup arrangement. We created workload by generating datasets with consistent, regular, left-skewed, and right-skewed distributions; these are, respectively, designated W01, W02, W03, and W04. W01, or a consistent workload, is made up of all tasks distributed equally. W02 is a workload with a regular distribution, meaning that there are more medium-sized jobs and less small, high-level tasks. There are more minor tasks and fewer large tasks in W03's left-skewed distribution of tasks. Lastly, W04's workload is skewed to the right, with a high proportion of major jobs and a low proportion of little ones. The simulation configuration settings are shown in Table 1 below (Mangalampalli et al., 2023). Additionally, we employed HPC2N's real-time logs, which are denoted by W05.

Table 1: Configuration Settings Used for Simulation

Name	**Quantity**
No. of Tasks	100-1000
Task Length	800,000
Memory of Physical Host	16 GB
Storage capacity of Physical Host	1 TB
Bandwidth Capacity	100 Mbps
Number of Virtual Machines	30
Memory capacity of virtual machine	1 GB
Bandwidth capacity of Virtual network	10 Mbps
No. of Processing elements	1050 MIPS
No. of Datacenters	5

The HPC2N Seth log refers to the logs generated by the High-Performance Computing Center North (HPC2N) on the Seth cluster. These logs are crucial for tracking the performance, usage, and any issues that arise within the cluster (HPC2N: the HPC2N Seth log) shows in figure 4.

JobID	SubmitTime	WaitTime	RunTime	Procs	UsedMem	ReqTime	ReqProcs	UserID	GroupID	Status	StartTime	EndTime
000001	250980	0	360	16	1000	360	16	14	1	1	250980	251340
000002	251000	10	600	32	2000	600	32	23	2	1	251010	251610
000003	251020	5	120	8	500	300	8	35	1	1	251025	251145
000004	251040	0	1800	64	4000	2000	64	41	2	1	251040	252840
000005	251060	20	720	16	1000	800	16	12	1	1	251080	251800

Figure 4: Snapshot of HPC2N Worklogs

The HPC2N workload is characterized by diverse task sizes and arrival patterns, making it an ideal benchmark for assessing the scheduler's efficiency.

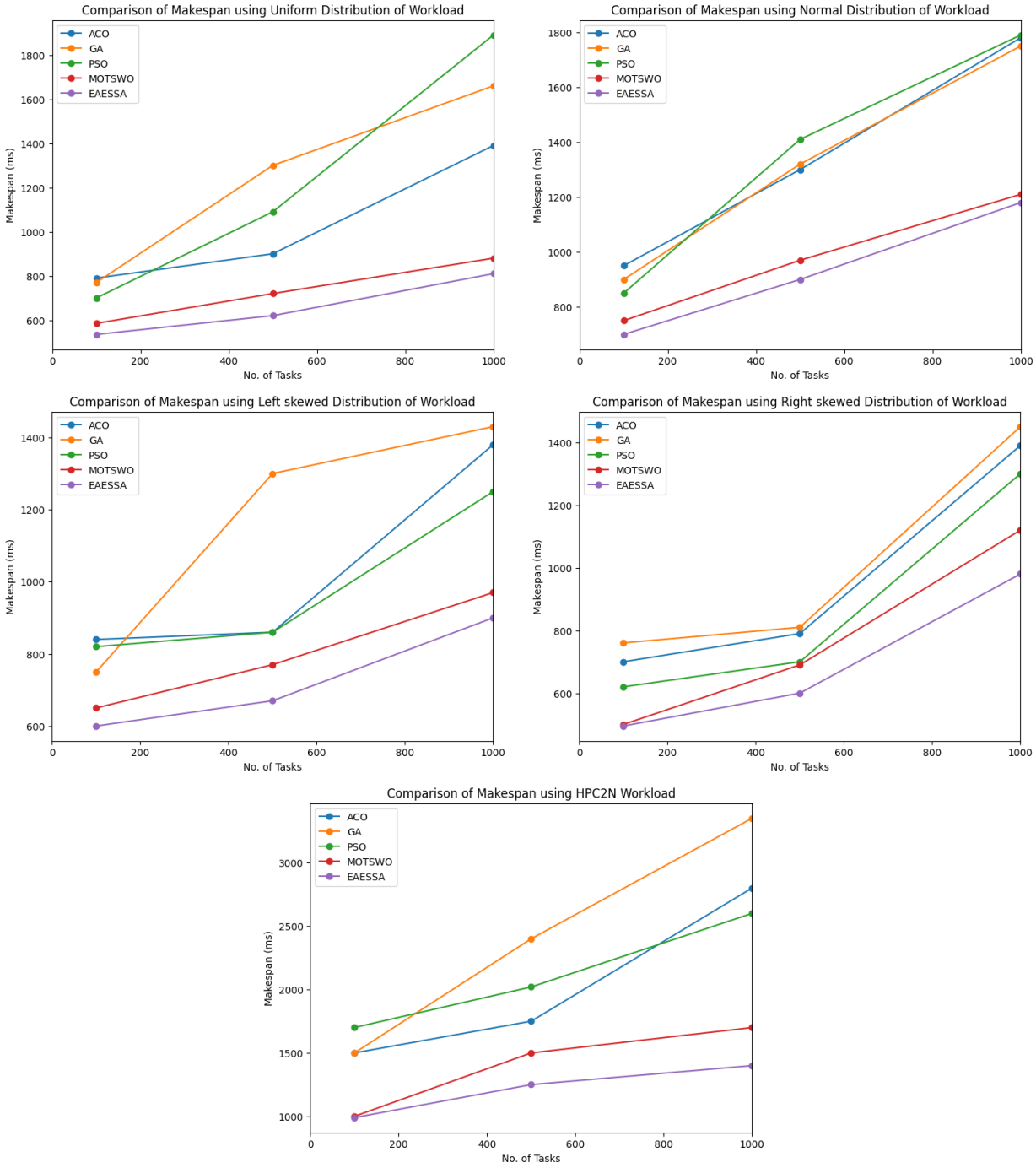

Figure 5: Comparison of Makespan

Estimation of Makespan

Makespen is calculated for the range of 100 to 1000 tasks and the simulations were run for 21 iterations. The arrangement shown in table one is used for W01, W02, W03, W04 and W05 respectively. The proposed algorithm EA-ESSA is assessed with four other existing algorithms i.e. ACO, GA, PSO and MOTSWAO approaches. From the Figure 5, it is clearly shows that EAESSA outperforms other approaches in makespan.

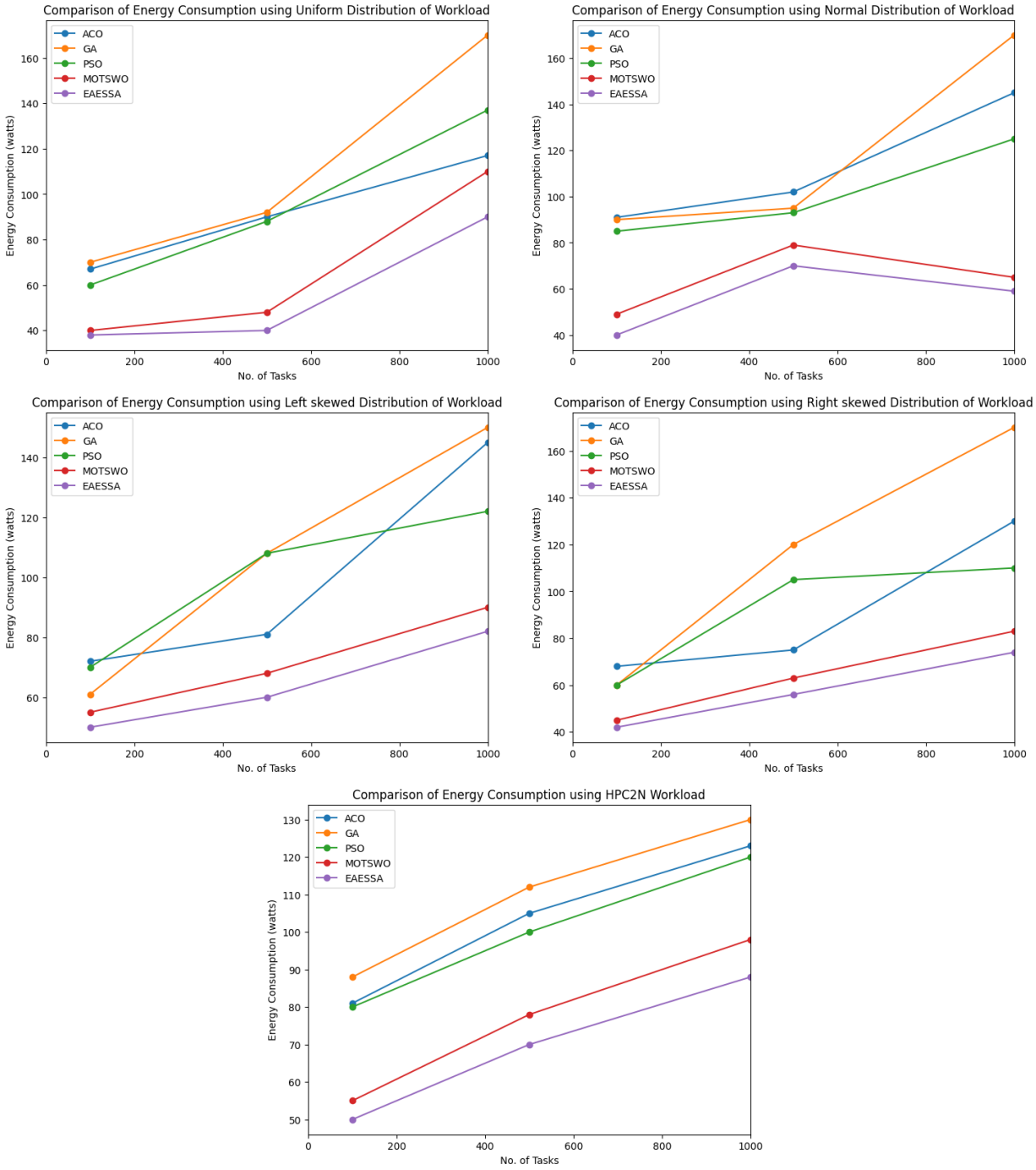

Figure 6: Comparison of Energy Consumption

Estimation of Energy Consumption

Energy Consumption is calculated for the range of 100 to 1000 tasks and the simulations were run for 21 iterations. The arrangement shown in table one is used for W01, W02, W03, W04 and W05 respectively. The proposed algorithm EA-ESSA is assessed with four other existing algorithms i.e. ACO, GA, PSO and MOTSWAO approaches. From the Figure 6, it is clearly shows that EAESSA has the minimum energy consumption compared to other approaches.

The suggested technique for makespan consumption has been assessed using Montage, Inspiral, CyberShake, Sipht, and Epigenomics, as shown in Figure 5, by differing the total cost. Figure 6 clearly shows that the suggested technique is effective at lowering energy usage regardless of the type of load. The proposed method outperforms in selecting the VMs for scheduling jobs and budget costs. The EA-ESSA algorithm outperformed baseline methods in cost management by integrating real-time spot pricing APIs.

7 Conclusion

The hybrid approach combines the communication minimization benefits of M-TDSA with the global resource optimization capabilities of M-SSA. Together, they provide an efficient and scalable solution for scheduling workflows in cloud environments, ensuring that tasks are executed with minimal delays, optimal resource usage, and reduced costs. This hybrid method is highly suitable for dynamic, large-scale cloud computing systems where task dependencies and resource availability are critical in performance optimization. This algorithm is well suited to integrate with current cloud frameworks within the user-defined financial parameters.

References

[1] Albtoush, A., Yunus, F., & Noor, N. M. M. (2024). Multi-priority scheduling algorithm for scientific workflows in cloud. *Bulletin of Electrical Engineering and Informatics*, *13*(4), 2979-2990. https://doi.org/10.11591/eei.v13i4.7520

[2] Alsadie, D., & Alsulami, M. (2024). Enhancing workflow efficiency with a modified Firefly Algorithm for hybrid cloud edge environments. *Scientific Reports*, *14*(1), 24675. https://doi.org/10.1038/s41598-024-75859-3

[3] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, *41*(1), 23-50. https://doi.org/10.1002/spe.995

[4] Canon, L. C., Chang, A. K. W., Robert, Y., & Vivien, F. (2020). Scheduling independent stochastic tasks under deadline and budget constraints. *The International Journal of High Performance Computing Applications*, *34*(2), 246-264. https://doi.org/10.1177/1094342019852135

[5] Cao, B., Zhao, J., Yang, P., Gu, Y., Muhammad, K., Rodrigues, J. J., & de Albuquerque, V. H. C. (2019). Multiobjective 3-D topology optimization of next-generation wireless data center network. *IEEE Transactions on Industrial Informatics*, *16*(5), 3597-3605.

[6] Chen, W., Xie, G., Li, R., Bai, Y., Fan, C., & Li, K. (2017). Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems. *Future Generation Computer Systems*, *74*, 1-11. https://doi.org/10.1016/j.future.2017.03.008

[7] Chintale, P., Malviya, R. K., Merla, N. B., Chinna, P. P. G., Desaboyina, G., & Sure, T. A. R. (2024, August). Levy Flight Osprey Optimization Algorithm for Task Scheduling in Cloud Computing. In *2024 International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS)* (pp. 1-5). IEEE. https://doi.org/10.1109/IACIS61494.2024.10721633

[8] Chithra Devi, S. A., Mahendravarman, I., Ragavendiran, A., Selvam, M. M., Yamuna, K., & Vibin, R. (2024, July). Optimal Configuration of Radial Distribution Networks with Stud Krill Herd Optimization. In *2024 International Conference on Signal Processing, Computation, Electronics, Power and Telecommunication (IConSCEPT)* (pp. 1-6). IEEE. https://doi.org/10.1109/IConSCEPT61884.2024.10627797

[9] El-Saadawi, E., Abohamama, A. S., & Alrahmawy, M. F. (2024). IoT-based optimal energy management in smart homes using harmony search optimization technique. *International Journal of Communication and Computer Technologies, 12*(1), 1-20. https://doi.org/10.31838/IJCCTS/12.01.01

[10] Fan, X., Weber, W. D., & Barroso, L. A. (2007). Power provisioning for a warehouse-sized computer. *ACM SIGARCH computer architecture news*, *35*(2), 13-23. https://doi.org/10.1145/1273440.1250665

[11] Fan, Y., Wang, L., Chen, J., Jin, Z., Shi, L., & Xu, J. (2020). Multi-job associated task scheduling based on task duplication and insertion for cloud computing. In *Wireless Algorithms, Systems, and Applications: 15th International Conference, WASA 2020, Qingdao, China, September 13–15, 2020, Proceedings, Part I 15* (pp. 109-120). Springer International Publishing. https://doi.org/10.1007/978-3-030-59016-1_10

[12] Ghafir, S., Alam, M. A., Siddiqui, F., Naaz, S., Sohail, S. S., & Madsen, D. Ø. (2024). Toward optimizing scientific workflow using multi-objective optimization in a cloud environment. *Cogent Engineering*, *11*(1), 2287303. https://doi.org/10.1080/23311916.2023.2287303

[13] Giliberto, M., Arena, F., & Pau, G. (2019). A fuzzy-based Solution for Optimized Management of Energy Consumption in e-bikes. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, *10*(3), 45-64. https://doi.org/10.22667/JOWUA.2019.09.30.045

[14] Goyal, A., Vyas, R. K., Rathi, K., & Singh, R. (2024, February). Task Scheduling Algorithm Improvement in Cloud Computing. In *2024 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)* (pp. 1-7). IEEE. https://doi.org/10.1109/SCEECS61402.2024.10482287

[15] Gupta, I., Kumar, M. S., & Jana, P. K. (2016, August). Task duplication-based workflow scheduling for heterogeneous cloud environment. In *2016 Ninth International Conference on Contemporary Computing (IC3)* (pp. 1-7). IEEE. https://doi.org/10.1109/IC3.2016.7880207

[16] Hilman, M. H., Rodriguez, M. A., & Buyya, R. (2020). Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions. *ACM Computing Surveys (CSUR)*, *53*(1), 1-39. https://doi.org/10.1145/3368036

[17] HPC2N: the HPC2N Seth log. https://www.cs.huji.ac.il/labs/parallel/workload/l_hpc2n/

[18] Hua, X., Jingling, Y., & Nana, W. (2022, May). A budget-constrained energy-efficient scheduling algorithm on cloud-edge collaborative workflows. In *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (pp. 432-437). IEEE. https://doi.org/10.1109/CSCWD54268.2022.9776086

[19] Kamalam, G. K., Shah, N. H., Krishnasamy, L., Pathak, L. K., & Vanitha, P. (2023, September). TSM: A Cloud Computing Task Scheduling Model. In *2023 6th International Conference on Contemporary Computing and Informatics (IC3I)* (Vol. 6, pp. 1575-1580). IEEE. https://doi.org/10.1109/IC3I59117.2023.10398151

[20] Mangalampalli, S., Karri, G. R., & Kose, U. (2023). Multi Objective Trust aware task scheduling algorithm in cloud computing using Whale Optimization. *Journal of King Saud University-Computer and Information Sciences*, *35*(2), 791-809. https://doi.org/10.1016/j.jksuci.2023.01.016.

[21] Mirjalili, S., Gandomi, A. H., Mirjalili, S. Z., Saremi, S., Faris, H., & Mirjalili, S. M. (2017). Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems. *Advances in engineering software*, *114*, 163-191. https://doi.org/10.1016/j.advengsoft.2017.07.002

[22] Muniyappa, V., & Hattibelagal, C. (2023). Cloud Computing for Task Scheduling Using Estimate of Distribution Algorithm-KrillHerd Method. *International Journal of Intelligent Engineering & Systems*, *16*(4). https://doi.org/10.22266/ijies2023.0831.49

[23] Prasath, C. A. (2024). Energy-efficient routing protocols for IoT-enabled wireless sensor networks. *Journal of Wireless Sensor Networks and IoT, 1*(1), 1-7. https://doi.org/10.31838/WSNIOT/01.01.01

[24] Rani, E. G., & Tukkoji, C. D. (2024). Secure Framework Optimizes QAES Technique Used for Computing in the Cloud. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 15*(3), 312-324. https://doi.org/10.58346/JOWUA.2024.I3.021

[25] Sadotra, P., Chouksey, P., Chopra, M., Koser, R., & Rawat, R. (2025). Research Review on Task Scheduling Algorithm for Green Cloud Computing. *Scalable Modeling and Efficient Management of IoT Applications*, 137-152. https://doi.org/10.4018/979-8-3693-1686-3.ch007

[26] Sandhu, R., Kaur, H., Sohal, H. S., Handa, K., Singh, A., & Islam, S. M. (2023, November). Enhancement in Cloud Performance using the Clustering Method of Scientific Workflow Tasks. In *2023 2nd International Conference on Ambient Intelligence in Health Care (ICAIHC)* (pp. 1-6). IEEE. https://doi.org/10.1109/ICAIHC59020.2023.10431461

[27] Saravanan, S., Manoharan, J. S., Kumar, V., & Surenderanath, S. (2022). A Dynamic Fuzzy Engine for adaptive control towards improvement of network performance in big data environment. *Procedia Computer Science*, *215*, 24-32. https://doi.org/10.1016/j.procs.2022.12.003

[28] Shi, L., Xu, J., Wang, L., Chen, J., Jin, Z., Ouyang, T., ... & Fan, Y. (2021). Multijob associated task scheduling for cloud computing based on task duplication and insertion. *Wireless Communications and Mobile Computing*, *2021*(1), 6631752. https://doi.org/10.1155/2021/6631752

[29] Stevovic, I., Hadrović, S., & Jovanović, J. (2023). Environmental, social and other non-profit impacts of mountain streams usage as Renewable energy resources. *Archives for Technical Sciences, 2*(29), 57-64. https://doi.org/10.59456/afts.2023.1529.057S

[30] Surendar, A. (2024). Emerging trends in renewable energy technologies: An in-depth analysis. *Innovative Reviews in Engineering and Science, 1*(1), 6-10. https://doi.org/10.31838/INES/01.01.02

[31] Tan, W., Sarmiento, J., & Rosales, C. A. (2024). Exploring the Performance Impact of Neural Network Optimization on Energy Analysis of Biosensor. *Natural and Engineering Sciences, 9*(2), 164-183. https://doi.org/10.28978/nesciences.1569280

[32] Ulabedin, Z., Khan, P., & Uddin, B. (2024). MCPF: Fault-Tolerant Scheduling of Scientific Workflow on Cloud Computing. https://doi.org/10.21203/rs.3.rs-4155303/v1

[33] Wang, Y., & Shi, W. (2014). Budget-driven scheduling algorithms for batches of MapReduce jobs in heterogeneous clouds. *IEEE Transactions on Cloud Computing*, *2*(3), 306-319. https://doi.org/10.1109/TCC.2014.2316812

[34] Wu, X., Cheng, S., Yuan, S., & Wang, Z. (2022, April). A Parallelism-Based Earliest Finish Time (PBEFT) Algorithm for Workflow Scheduling in Clouds. In *2022 7th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)* (pp. 22-26). IEEE. https://doi.org/10.1109/ICCCBDA55098.2022.9778917

[35] Xue, S., Li, M., Xu, X., Chen, J., & Xue, S. (2014). An ACO-LB Algorithm for Task Scheduling in the Cloud Environment. *J. Softw.*, *9*(2), 466-473.

[36] Yao, F., Pu, C., & Zhang, Z. (2021). Task duplication-based scheduling algorithm for budget-constrained workflows in cloud computing. *IEEE Access*, *9*, 37262-37272. https://doi.org/10.1109/ACCESS.2021.3063456

[37] Zhang, L., Wang, L., Wen, Z., Xiao, M., & Man, J. (2020). Minimizing energy consumption scheduling algorithm of workflows with cost budget constraint on heterogeneous cloud computing systems. *IEEE Access*, *8*, 205099-205110. https://doi.org/10.1109/ACCESS.2020.3037205

[38] Zhang, R., Tian, F., Ren, X., Chen, Y., Chao, K., Zhao, R., ... & Wang, W. (2018). Associate multi-task scheduling algorithm based on self-adaptive inertia weight particle swarm

optimization with disruption operator and chaos operator in cloud environment. *Service Oriented Computing and Applications*, *12*, 87-94. https://doi.org/10.1007/s11761-018-0231-7

[39] Zigui, L., Caluyo, F., Hernandez, R., Sarmiento, J., & Rosales, C. A. (2024). Improving Communication Networks to Transfer Data in Real Time for Environmental Monitoring and Data Collection. *Natural and Engineering Sciences*, *9*(2), 198-212. https://doi.org/10.28978/nesciences.1569561

Authors Biography

C.K. Sripavithra received her B.Sc. in Applied Science-Computer Technology and MCA from Bharathiar University. She received her Master's in philosophy from Periyar University. She possesses two decades of experience in teaching and is currently serving as an Assistant Professor at Maharani's Science College for Women (Autonomous), Mysore affiliated to the University of Mysore, Mysore, India. Currently, she is pursuing research in the area of cloud computing from Christ (Deemed to be University), India under the supervision of Dr.V.B. Kirubanand. She has presented her research articles in many National & International Conferences and has 10 publications to her credit.

Dr.V.B. Kirubanand is currently working as Associate Professor at CHRIST (Deemed to be University), Bangalore. He completed his undergraduate in Electronics and Masters in Computer Application from Bharathiar University, Coimbatore. He received his Master's in philosophy from Periyar University, Salem and Doctoral degree from Anna University, Chennai. He possesses two decades of experience in teaching and expertise in Wireless Networks, Web Technology and Internet of Things. He has 13 book publications to his credit and has a good track record of 77 research publications in Scopus and high impact factor journals. 8 Patents published & 4 Design Granted. He has organised several National and International conferences, seminars and webinars.