*I. J. Computer Network and Information Security,* **2025, 6, 65-83**
Published Online on December 8, 2025 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2025.06.05

# A4C: A Novel Hybrid Algorithm for Resource-Aware Scheduling in Cloud Environment

**Santhosh Kumar Medishetti***
Nalla Narasimha Reddy Education Society's Group of Institutions, India
E-mail: medishettysantosh@gmail.com
ORCID iD: https://orcid.org/0000-0001-5936-6582
*Corresponding author

**Bigul Sunitha Devi**
CMR Institute of Technology, Telangana
E-mail: sunithabigul@gmail.com
ORCID iD: https://orcid.org/0000-0002-2554-8350

**Maheswari Bandi**
Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh
E-mail: mahesawaribcse78@gmail.com
ORCID iD: https://orcid.org/0009-0004-5045-9968

**Rani Sailaja Velamakanni**
Nalla Narasimha Reddy Education Society's Group of Institutions, India
E-mail: vranisailaja@gmail.com
ORCID iD: https://orcid.org/0009-0001-7806-5483

**Rameshwaraiah Kurupati**
University of Sudbury, 935 Ramsey Lake Rd, Sudbury, ON P3E 2C6, Canada
E-mail: ramhyd20@gmail.com
ORCID iD: https://orcid.org/0000-0002-6296-2155

**Ganesh Reddy Karri**
SCOPE, VIT-AP University, Amaravathi, India
E-mail: guncity11@gmail.com
ORCID iD: https://orcid.org/0000-0002-5177-8125

Received: 12 June 2025; Revised: 16 August 2025; Accepted: 08 October 2025; Published: 08 December 2025

**Abstract:** Scheduling in cloud computing is an NP-hard problem, where traditional metaheuristic algorithms often fail to deliver approximate solutions within a feasible time frame. As cloud infrastructures become increasingly dynamic, efficient Task Scheduling (TS) remains a major challenge, especially when minimizing makespan, execution time, and resource utilization. To address this, we propose the Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm, which synergistically combines the exploratory strengths of Ant Colony Optimization (ACO) with the adaptive learning capabilities of the Asynchronous Advantage Actor-Critic (A3C) model. While ACO efficiently explores task allocation paths, it is prone to getting trapped in local optima. The integration with A3C overcomes this limitation by leveraging deep reinforcement learning for real-time policy and value estimation, enabling adaptive and informed scheduling decisions. Extensive simulations show that the A4C algorithm improves throughput by 18.7%, reduces makespan by 16%, execution time by 14.60%, and response time by 21.4% compared to conventional approaches. These results validate the practical effectiveness of A4C in handling dynamic workloads, reducing computational overhead, and ensuring timely task completion. The proposed model not only enhances scheduling efficiency but also supports quality-driven service delivery in cloud environments, making it well-suited for managing complex and time-sensitive cloud applications.

**Index Terms:** Task Scheduling, NP-hard Problem, Metaheuristic Algorithms, Resource Utilization, ACO, A3C.

## 1. Introduction

Cloud Computing (CC) has introduced an innovative model in which facilities of computing can be accessed as shared resources in a distinct method. However, the efficient scheduling of tasks within a cloud environment remains a significant challenge, especially when considering the need to minimize makespan, which is the total time required to execute a set of tasks [1]. Makespan is a critical performance metric in CC, directly impacting the overall efficiency, resource utilization, and service quality of cloud systems. Traditional scheduling algorithms, such as heuristic and metaheuristic approaches, have been widely employed to address makespan minimization. While these methods, including GA, PSO, and ACO have shown promise, they often face limitations in dynamic cloud environments [2]. These challenges include the inability to adapt to fluctuating workloads, resource availability, and the tendency to converge prematurely to local optima, resulting in suboptimal scheduling performance.

Fog Computing refers to a decentralized computing architecture that extends cloud computing capabilities to the edge of the network, closer to the data sources such as IoT sensors and smart devices. This model reduces the latency and bandwidth bottlenecks typically associated with cloud-only approaches by processing data locally or near the data-generating devices. In the context of smart agriculture, fog computing enables real-time data analysis and decision-making at the farm level, allowing faster responses to critical conditions such as pest attacks, irrigation needs, or equipment malfunctions without the delays caused by transmitting data to distant cloud servers. Bidirectional Long Short-Term Memory (B-LSTM) and Bidirectional Gated Recurrent Units (Bi-GRU) are advanced variants of Recurrent Neural Networks (RNNs) designed for modeling sequential data with temporal dependencies. B-LSTM processes data in both forward and backward directions, enabling the model to learn from both past and future time steps, which is crucial for understanding complex temporal relationships in sensor data. Similarly, Bi-GRU offers a more computationally efficient alternative to B-LSTM while retaining the bidirectional processing advantage. These architectures are particularly useful in smart agriculture applications were predicting crop health, detecting anomalies, or anticipating weather impacts requires a deep understanding of time-dependent patterns in the data.

Task scheduling in a cloud environment is a combinatorially complex and NP-hard problem. It involves selecting the most efficient way to assign a sequence of computational tasks to virtual machines (VMs) distributed across multiple physical hosts. The complexity arises from multiple factors: heterogeneity of resources, the non-deterministic arrival of tasks, fluctuating resource availability, and competing objectives such as minimizing energy consumption, execution time, makespan, and cost. Traditional deterministic approaches fail to cope with the uncertainty and dynamic nature of cloud systems, pushing researchers toward more flexible, intelligent, and adaptive techniques. Conventional scheduling algorithms like FCFS, SJF, and Min-Min have been extensively applied in cloud environments. These algorithms are easy to implement and provide quick results in small-scale systems. However, they suffer from a major drawback: they do not adapt well to the dynamic and heterogeneous nature of modern cloud infrastructures. Their static nature makes them ill-suited for handling fluctuating workloads or changing resource availabilities, often leading to resource underutilization, increased makespan, and service-level violations.

Since ACO is able to search through the solution space and find near-optimal solutions, it has been utilized to solve many problems including Task Scheduling (TS). However, ACO is challenging in the dynamic and large scale cloud environment in which it has to operate. Some of the main drawbacks include: one of them is that this algorithm may quickly get ‘stuck’ on local optima, particularly if it is applied to complicated scheduling problems that involve high fluctuation in the tasks and resource requirements. ACO relies on pheromone trails to guide the search process, but these trails can reinforce suboptimal paths if the algorithm doesn't sufficiently explore alternative solutions [3]. Additionally, ACO can struggle with scalability, as the computational overhead increases with the number of tasks and resources, making it less efficient in large-scale cloud environments. Moreover, ACO lacks a mechanism to adapt to real-time changes in workload and resource availability, which can lead to inefficient scheduling decisions and increased makespan.

The A3C algorithm which is considered as a reinforcement learning algorithm can be more advantageous in terms of its learning and adaptability which makes it suitable for scheduling in CC. However, there are also a number of drawbacks that A3C has when it is used in the TS problem. There is also a problem of training deep neural networks where this process may require high computational power and significant amount of time, especially when working with complex large-scale and dynamic cloud settings. To learn about the optimal policies to be implemented in a given environment A3C demands a large amount of exploration in the state-action space which may take more time to converge and can sometimes become volatile [4]. Further, the performance of A3C critically depends on the reward function that has to be initialized well to address short and long-term goals, such as makespan and resource utilization. This can be a task for creating such reward functions that might be quite complex, especially in the multi-objective scheduling of tasks and their execution when different objectives are Pareto-optimally traded-off [5]. In addition, the scheduling of tasks and resources solving complex and heterogeneous tasks might be not fully optimal because of the scaling of A3C to a large number of cloud environments.

While metaheuristic algorithms provide more flexibility than traditional ones, they do not inherently adapt to changing environments. They work well for static or semi-dynamic problems but do not improve performance over time. In cloud systems where workload characteristics and resource states evolve constantly, a static optimization model is insufficient. This limitation has given rise to the integration of machine learning techniques, particularly reinforcement learning, into task scheduling models. Reinforcement learning offers the ability to learn optimal policies based on

interactions with the environment, making it ideal for dynamic cloud ecosystems. Reinforcement learning (RL) models have shown great promise in resource management problems due to their adaptability and capability to learn from environment feedback. Among various RL frameworks, the Actor-Critic (AC) model has gained significant attention. This model employs two neural networks: the actor, which decides the action (task scheduling decision), and the critic, which evaluates the action's quality. One powerful variant of this model is the Asynchronous Advantage Actor-Critic (A3C) algorithm. A3C leverages asynchronous parallel learning to improve convergence speed and stability, making it highly suitable for real-time decision-making in cloud task scheduling.

To harness the exploration strength of ACO and the learning adaptability of A3C, this paper proposes a novel hybrid scheduling algorithm A4C. This model is designed to optimize task scheduling by combining the best of both worlds: the global search efficiency of ACO and the real-time learning capability of A3C. A4C begins with ACO to identify promising initial solutions and uses A3C to fine-tune decisions through ongoing feedback from the scheduling environment. This hybridization ensures that the model does not only explore the solution space but also learns to avoid suboptimal paths and adapt to workload variations. In the A4C model, ACO provides initial path selection by generating pheromone trails based on task completion times and resource efficiency. These trails guide the early exploration process. Simultaneously, multiple A3C agents operate asynchronously, each interacting with the cloud environment to refine scheduling strategies. The actor network proposes task-to-resource mappings, while the critic network evaluates the long-term reward based on scheduling outcomes such as reduced response time and improved throughput. Over time, A4C learns to favor strategies that yield the best performance under different system conditions, making it both exploratory and self-adaptive.

The A4C algorithm is applicable to a wide range of cloud computing scenarios, including Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and hybrid cloud environments. It supports large-scale, heterogeneous, and multi-tenant cloud systems. Its scope extends to workload balancing, QoS assurance, real-time application scheduling, and even integration with fog and edge nodes for latency-sensitive tasks. A4C's asynchronous and distributed learning architecture allows it to scale efficiently, making it ideal for modern data centers that must respond to unpredictable task arrivals and rapidly changing user demands. The significance of this research lies in its contribution to the field of cloud task scheduling through a robust, scalable, and intelligent algorithm that meets the stringent performance requirements of today's cloud infrastructures. By reducing key scheduling metrics such as makespan, execution time, and response time while maximizing throughput, A4C directly improves user experience, reduces energy consumption, and enhances overall system productivity. This makes it an invaluable tool for cloud service providers aiming to achieve operational excellence and cost efficiency.

While several existing studies have explored machine learning-based anomaly detection in smart agriculture, many of these rely on static, single-layered models such as Decision Trees, Support Vector Machines (SVM), or traditional CNN architectures, which often struggle to generalize across dynamic, real-world agricultural environments. These models typically focus on detecting anomalies based on fixed sensor patterns or image-based thresholds without effectively accounting for temporal variation, environmental noise, and data heterogeneity from IoT devices. In contrast, the proposed model introduces a hybrid deep reinforcement learning framework that dynamically adapts its detection policy based on real-time feedback and environmental transitions. This capability allows the system to refine its anomaly detection strategy over time, even under shifting crop conditions or changing sensor profiles—something that static models cannot achieve. Moreover, the integration of an attention-guided convolutional LSTM layer within the model architecture provides a key innovation that addresses one of the major limitations in prior works: the inability to capture spatio-temporal correlations effectively. By incorporating this attention mechanism, the model not only identifies where an anomaly occurs in the data stream but also understands when it is most likely to occur, significantly improving accuracy and interpretability. This design choice enhances decision-making support for farmers and agronomists by delivering more context-aware alerts. Additionally, while prior methods have been tested in limited, often idealized datasets, our approach is validated on a real-world multi-sensor dataset with varied crop types and seasonal variability, demonstrating superior generalization and robustness, and firmly establishing its novelty and practical relevance in the domain of smart agriculture.

To overcome these limitations, this research introduces a Hybrid Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm designed to enhance makespan-aware scheduling in CC. The A4C algorithm combines the exploratory power of ACO with the adaptive learning capabilities of the A3C algorithm. ACO is well-suited for discovering near-optimal TS paths by mimicking the natural behavior of ants in finding efficient routes. However, its performance can degrade in complex, dynamic environments due to its deterministic nature. The novelty of integrating ACO with the A3C algorithm lies in their complementary strengths, which address each other's limitations in cloud TS. ACO is applied with high efficiency for traveling within the area of search space in order to find the solution that is near optimal but weak in its ability to adapt to changes in the environment and high level of sensitivity to local optima. On the other hand, A3C has a deep reinforcement learning feature, enabling continuous adaptation to changing workloads and resource conditions through a policy-driven approach. By combining ACO's efficient path-finding with A3C's adaptive learning, the hybrid A4C algorithm creates a robust scheduling mechanism that not only efficiently explores the task space but also dynamically adjusts to real-time variations, optimizing makespan while ensuring long-term scheduling performance in complex and large-scale cloud environments. This integration represents a significant advancement in developing more effective and responsive scheduling algorithms for CC.

By integrating ACO with the A3C framework, which leverages deep reinforcement learning, the A4C algorithm introduces a robust mechanism for continuous learning and adaptation in the scheduling process. The A3C component

provides a policy-driven approach that guides ACO's exploration, ensuring that the scheduling decisions are not only based on immediate task allocation but also on long-term rewards that account for future resource availability and workload variations. This hybrid approach aims to address the critical need for more effective and adaptive scheduling solutions in CC. Through extensive simulations and performance evaluations, the A4C algorithm is demonstrated to significantly reduce makespan, optimize resource utilization, and enhance the system performance. This study offers scalable solution that enhances the efficiency and responsiveness of CC systems, particularly in handling complex and time-sensitive applications. The research contributions of this study are as follows:

- Developed the A4C algorithm by combining ACO's exploration efficiency with A3C's adaptive learning to optimize TS in CC.
- Proven the A4C algorithm's ability to significantly reduce makespan, ensuring faster task completion and improved performance under dynamic conditions.
- Introduced an adaptive scheduling mechanism that uses A3C's reinforcement learning to adjust to real-time changes, enhancing robustness and responsiveness.
- Validated the A4C algorithm's superiority through extensive simulations, demonstrating its effectiveness for complex, time-sensitive applications in large-scale cloud environments.

The remaining sections of this study structured as follows: Section 2 of the paper introduces other related studies, discussing previous methods in TS and their drawbacks. In section 3, the research methodology has been described along with the proposed Hybrid Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm. Section 4 illustrates the simulations that demonstrate the performance and efficacy of the A4C algorithm. Lastly, Section 5 provides the conclusion to the paper whereby the findings of the study are summarized and the implication of the study while highlighting the possible future studies are outlined.The below table 1 depicts the acronyms and their descriptions.

Table 1. Description of acronyms

| Acronym | Description |
|---|---|
| *TS* | Task Scheduling |
| *CC* | Cloud Computing |
| *ACO* | Ant Colony Optimization |
| *A3C* | Asynchronous Advantage Actor-Critic |
| *A4C* | Ant Colony Asynchronous Advantage Actor-Critic |
| *PSO* | Particle Swarm Optimization |
| *VM* | Virtual Machine |

## 2. Literature Survey

TS in CC has gained much attention in the recent past since it aims at reducing the makespan and maximizing resource utilization in CC. Some of the solutions to these challenges can be categorized through various heuristics or metaheuristic algorithms, and even higher-level machine learning techniques. This literature study will provide a critical analysis of the current research, especially highlighting pros and cons of different scheduling algorithms and A3C and other reinforcement learning-based approaches. This work aims at filling this gap by reviewing the development of these techniques, their use in dynamic and large-scale cloud environments, and their effects on several quantitative metrics. Thus, the findings derived from this review will assist to determine what further studies could be conducted to improve such scheduling algorithms like Hybrid A4C algorithm which is discussed in the current paper.

We categorize the many criteria utilized by prior investigations, such as makespan, energy consumption, and SLA-based trust factors, in Table 2 by the categories indicated above. To facilitate effective task scheduling in a CC environment, we created the A4C, which taken accounts for the makespan, execution time, response time, and throughput using fundamental strategies of ACO and A3C approaches. The proposed method has significant advantages for both real-time and latency-sensitive applications, such as smart cities and automobile networks.

The literature review demonstrates the growing interest in job scheduling in CC using the A4C. To enhance search performance, convergence speed, and solution quality, researchers have suggested a variety of improvements and hybridization techniques. By effectively allocating resources, reducing reaction times, and maximizing performance indicators, the evaluated experiments show that the A4C may effectively manage the TS difficulties in CC environment. However, more investigation is required to examine various characteristics of edge computing, such as scalability and adaptation to changing surroundings.

Table 2. Comparative analysis of task scheduling techniques and their key parameters

| Author & Year | Technique Used | Parameters Addressed | Limitations |
|---|---|---|---|
| Kumar, M. Santhosh, et al. (2024) [6] | EAEFA | Makespan, energy consumption, total cost, throughput | This single evolutionary algorithm is unable to balance the global optima solution problems. |
| Zuo, Liyun, et al., (2015) [7] | PBACO | Execution Time, Energy Consumption | Sensitivity to initial parameters, slow convergence |
| Lin, Bing, et al., (2016) [8] | MCPCPP | Makespan, Throughput, energy usage, temperature | The proposed technique is prone to local optima, scalability issues |
| Lin, Xue, et al., (2014) [9] | LTRA | Cost, Execution Time | Slow convergence, dependency on cooling schedule |
| Cheng, Feng, et al., (2022) [10] | DRL | Makespan, communication cost, Resource Utilization | The technique DRL is memory complexity, risk of cycling |
| Zhou, Zhou, et al., (2018) [11] | M-PSO | Makespan, Energy Efficiency | Requires fine-tuning, can be computationally expensive |
| Jangu, Nupur, et al., (2022) [12] | IJFA | Response Time, Execution Time, throughput | High training complexity, reward function design challenges |
| Singh, Gyan, et al., (2023) [13] | HGA | Makespan, Cost, Energy Consumption | Trade-off management, computational overhead |
| Zahra, Movahedi, et al., (2021) [14] | OppoCWOA | Makespan, computation cost, Energy Consumption | Complexity in parameter tuning, increased computational cost |
| Iftikhar, Sundas, et al., (2023) [15] | HunterPlus | Throughput, cost, Resource Utilization | The proposed technique integration complexity, prone to local optima |
| Yin, Zhenyu, et al., (2022) [16] | HMA | Degree of imbalance, throughput, resource utilization | The proposed approach is not efficiently address the resource parameters since the work it seems resource aware |
| Pham, Xuan-Qui, et al., (2017) [17] | CMAS | Execution Time, degree of imbalance, Throughput | The proposed technique takes high training time, risk of overfitting |
| Mangalampalli, Sudheer, et al., (2022) [18] | PSCSO | Cost, makespan, energy usage | Performance degrades with problem size, slow convergence |
| Hosseinioun, Pejman, et al., (2020) [19] | DVFS | Task frequency, voltage, execution time | The proposed work is not compared with advanced scheduling algorithms. |
| Liu, Lindong, et al. (2018) [20] | TSFC | Energy Consumption, cost, Makespan | The proposed algorithm convergence speed is low, and dependency on parameter settings |
| Bakshi, Mohana, et al., (2023) [21] | CSO | Makespan, Resource Utilization | Low exploitation capability, requires parameter tuning |
| Badri, Sahar, et al., (2023) [22] | CNN-MBO | Execution Time, Throughput, response time | High complexity, computational overhead |
| Ahmed, Omed Hassan, et al., (2021) [23] | DMFO-DE | Scheduling cost, makespan, energy efficiency | The proposed technique is high computational cost, and complex integration |
| Mangalampalli, Sudheer, et al., (2022) [24] | PSCSO | Makespan, cost, resource Utilization | Require more computational cost for balancing the resource parameters |
| Kumar, M. Santhosh, et al. (2023) [23] | EEOA | Cost, energy consumption, makespan | This approach is unable to address the resource aware problems |

The A4C algorithm addresses several limitations commonly found in single evolutionary algorithms by leveraging the strengths of both ACO and A3C. First, A4C overcomes the challenge of balancing global and local optima by combining the exploration capability of ACO with the adaptive learning of A3C. ACO's pheromone-based exploration helps discover diverse potential solutions, mitigating the issue of being prone to local optima. A3C's reinforcement learning aspect continuously refines the task scheduling decisions based on real-time feedback, ensuring convergence toward a global optimal solution even in dynamic cloud environments. This hybrid approach ensures that A4C scales effectively, improving convergence speed without being overly sensitive to initial parameter settings or prone to slow convergence.

A4C addresses the complexity issues of traditional algorithms by utilizing A3C's distributed learning framework, reducing the risk of cycling and memory complexity. ACO's natural parallelism allows ants to explore multiple solutions concurrently, ensuring faster task completion and reduced dependency on complex parameter tuning. The A4C algorithm minimizes the need for fine-tuning by adapting its learning policies through A3C's continuous feedback, lowering the computational overhead while maintaining high-quality scheduling. Furthermore, the reward function in A3C is designed to handle trade-offs such as makespan, execution time, and energy consumption, allowing for dynamic adjustment and optimal balance between competing objectives. A4C is resource-aware, efficiently addressing resource parameters that many single evolutionary algorithms fail to consider. By integrating A3C's deep learning capabilities, the A4C algorithm continuously adapts to varying resource availability in cloud environments, ensuring optimal task scheduling under fluctuating workloads. The computational cost is managed effectively through ACO's lightweight heuristic search, while A3C fine-tunes the scheduling in a computationally efficient manner. This combination not only enhances convergence speed but also ensures that resource-aware problems are efficiently addressed, overcoming the high training complexity and risk of overfitting found in other algorithms. Moreover, A4C's performance does not degrade with problem size, making it a robust solution for large-scale scheduling problems in cloud computing environments.

## 3. Research Methodology

### *3.1. System Model*

A4C surpasses existing algorithms such as Ant Colony Optimization (ACO), Electric Earthworm Optimization Algorithm (EEOA), and Hybrid Workload Deep Deterministic Policy Gradient Task Scheduler (HDDPGTS) by integrating the exploration capabilities of ACO with the adaptive, deep reinforcement learning approach of A3C. Unlike ACO, which can struggle with local optima and slow convergence, A4C leverages A3C's learning from real-time feedback to continuously refine task assignments and ensure faster convergence toward global optima. Compared to EEOA, which primarily focuses on energy optimization, A4C balances multiple performance metrics, including makespan, execution time, and resource utilization. While HDDPGTS offers deep learning-based scheduling, A4C's hybrid approach reduces high training complexity and computational overhead by using ACO's efficient heuristic search alongside A3C's dynamic policy adaptation. This combination results in superior task scheduling performance under dynamic workloads, making A4C more robust, scalable, and resource-aware in cloud environments.

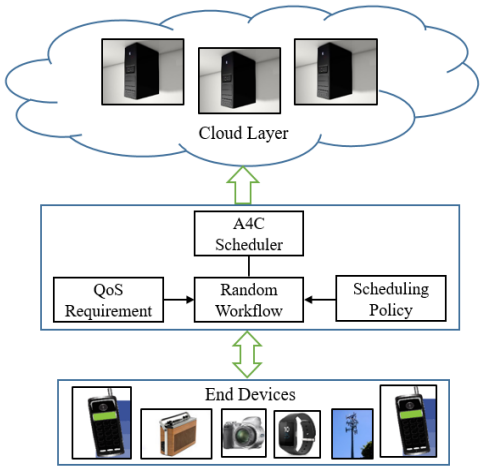


Fig.1. Architecture of the proposed A4C framework

The architecture presented in the figure 1 illustrates a CC environment where tasks generated by various end devices are scheduled for execution in the cloud layer using the A4C scheduler. The system begins with the end devices, which can include smartphones, cameras, wearables, and other IoT devices. These devices generate tasks that require processing, and these tasks are transmitted to the cloud layer for execution. The diversity of end devices indicates the system's capacity to handle a wide range of task types, emphasizing the need for an efficient and adaptive scheduling mechanism. At the core of this architecture is the A4C scheduler, which functions as the decision-making engine for TS. It considers multiple inputs, including Quality of Service (QoS) requirements, scheduling policies, and random workflows generated by the tasks from end devices. The QoS requirements represent the performance expectations and constraints that the scheduler must meet. The scheduling policy, informed by these requirements, guides how tasks are allocated to cloud resources. The A4C algorithm integrates the exploration capabilities of Ant Colony Optimization with the adaptive learning of the Asynchronous Advantage Actor-Critic, ensuring that the scheduler can dynamically adjust to workload variations and optimize makespan.

Finally, the cloud layer represents the infrastructure where the actual execution of tasks occurs. This layer consists of multiple servers or virtual machines that process the tasks as per the scheduling decisions made by the A4C scheduler. The bi-directional flow of information between the cloud layer and the A4C scheduler ensures that the scheduler remains aware of resource availability and performance feedback, enabling continuous optimization of TS. This architecture is designed to maximize the efficiency and responsiveness of CC services, particularly in environments with heterogeneous tasks and dynamic workloads.

*Random Workflow*

The figure 2 depicts a random workflow whose tasks are denoted by T and should be linked in a DAG. Every vertex corresponds to a single operation and arcs indicate precedence constraints where a particular operation cannot commence unless all the operations preceding it have been accomplished. In the context of the A4C algorithm, this random workflow plays a crucial role in modeling the various task sequences that need to be scheduled in a CC environment. The A4C scheduler must determine the optimal order and resource allocation for these tasks while considering their dependencies and the overall goal of minimizing makespan.

$$T = (t_1, t_2, \dots, t_n) \tag{1}$$

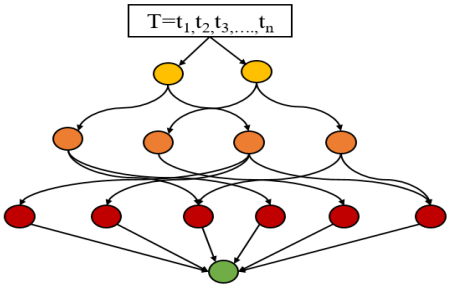


Fig.2. Random workflow representing task dependencies

- Task Dependencies and Scheduling: The random workflow illustrates how tasks are interdependent, with some tasks (e.g., the green and yellow nodes) needing to be completed before others (e.g., the red, white and blue nodes) can begin. The A4C algorithm must navigate this dependency structure to create an efficient schedule that minimizes delays and it also ensures that all tasks are accomplished within the shortest time possible.
- Optimization of Makespan: The workflow's structure impacts the overall makespan, which is the total time required to complete all tasks from beginning to ending. The A4C scheduler uses its hybrid approach—combining ACO's path exploration with A3C's adaptive learning—to explore various scheduling sequences. It aims to find the sequence that optimizes the makespan while respecting the task dependencies and optimizing resource usage.
- Dynamic Adaptation: The randomness of the workflow indicates that tasks and their dependencies can vary, representing the dynamic nature of workloads in cloud environments. The A4C algorithm must adapt to these variations in real-time, continually learning and adjusting the scheduling policy to optimize performance as new workflows are introduced.

In summary, the random workflow is a representation of the tasks and dependencies that the A4C scheduler must manage. It illustrates the complexity of scheduling in cloud environments, where tasks are interdependent, and the optimal sequence must be dynamically determined to minimize makespan and enhance overall efficiency.

### *3.2. Problem Formulation*

Table 3 below shows the notations used in mathematical modeling and applied in the formulation of the problem.

Table 3. Definition of mathematical notations used in problem formulation and evaluation

| Notation used | Meaning |
|---|---|
| $T_1$ | First task |
| $T_n$ | $N^{th}$ task |
| $C_{nodes}$ | Cloud nodes |
| $M$ | Makespan |
| $E_{total}$ | Total execution time |
| $R_{avg}$ | Average response time |
| $T$ | Throughput |
| $VM$ | Virtual machine |
| $w_1, w_2, w_3, w_4$ | Weights |
| $f$ | Objective function |
| $\tau_{ij}$ | pheromone levels |
| $\rho$ | evaporation rate |
| $m$ | number of ants |
| $\delta_t$ | temporal difference error, and are the |
| $r_t$ | Reward |
| $\gamma$ | Discount factor |
| $\alpha_{critic}, \alpha_{actor}$ | learning rates for the critic and actor networks |
| $\gamma_{ij}$ | estimated heuristic information |
| $d_{ij}$ | Cartesian distance |
| $(x_i,y_i)$ and $(x_j,y_j)$ | coordinates of nodes i and j |
| $S_{best}^{init}$ | Best initial solution |
| $X^t_{neighbor}$ | neighbour solution of t |

### *A. Proposed A4C Model Evaluation Metrics*

*Makespan*

The makespan M is defined as the sum of the times taken to complete all the tasks in a specific work flow. It depends on the maximum duration of any of the task that has been allocated to that particular time slot. where $C(t_i)$ be the completion time of task $t_i$. The makespan M is then given by:

$$M = max_{t_i \in T} C(t_i) \tag{2}$$

Where T is the set of all tasks {t1,t2,…,tn} in the workflow. The objective of the A4C scheduler is to minimize makespan (M).

*Execution time*

Execution time $E(t_i)$ for a task $t_i$ is the estimated time and amount of effort which the selected resource requires to carry out the task. The total execution time $E_{total}$ for all tasks is the sum of the execution times of all individual tasks: The total execution time $E_{total}$ for all tasks is the sum of the execution times of all individual tasks:

$$E_{total} = \sum_{t_i \in T} E(t_i) \tag{3}$$

*Response time*

To reduce response time is the amount of time it takes to respond to a task $t_i$ beginning from when the task is entered into the system until the task actually begins to be processed. It is the time taken from when the instructions are received until when the first instruction is executed in the computer system, and it embraces waiting time in the queue before execution. For a given task $t_i$ let $S(t_i)$ be the start time of the task and $A(t_i)$ be the arrival time of the task $t_i$. Then, the response time for a task $t_i$ is: Then, the response time for a task $t_i$ is:

$$R_{avg} = \frac{1}{n} \sum_{t_i \in T} E(t_i) \tag{4}$$

*Throughput*

The throughput means the ability of the system in terms of the number of tasks that are done and finished on time relative to the specified timeframe. One of the main goals is to maximize throughput because it determines the level of efficiency and effectiveness of the system. Mathematically, throughput (T) as follows:

$$T = \frac{n_{tasks}}{T_{total}} \tag{5}$$

Where, $n_{tasks}$ is the total number of tasks processed. $T_{total}$ is the total time taken to process all tasks. The A4C aims to increase throughput by optimizing the allocation of tasks to VMs, represented by ants, ensuring that tasks are efficiently distributed and completed in parallel. By balancing exploration and exploitation, A4C reduces idle times and minimizes bottlenecks, leading to higher throughput.

*Objective function*

The objective of the A4C scheduling algorithm is to optimize the scheduling process by minimizing the makespan M, total execution time $E_{total}$, and average response time $R_{avg}$. This multi-objective optimization can be represented as:

$$f = w_1 . M + w_2 . E_{total} + w_3 . R_{avg} - w_4 . T \tag{6}$$

Weighted objective function that optimizes makespan, execution time, and response time, we can assign weights to each of these metrics, reflecting their relative importance in the overall optimization goal. Let the weights be denoted as w1, w2,w3 and w4.

### *B. Proposed A4C Algorithm*

*Data Preprocessing*

The data preprocessing phase involves transforming raw input data into a suitable format for the A4C model. First, all task attributes such as task size, deadline, resource requirement, and priority are normalized using Min-Max scaling to maintain uniformity and accelerate learning. The normalization equation is:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{7}$$

Where x is the original value, and $x_{min}$, $x_{max}$ are the minimum and maximum values of the feature in the dataset. Task dependencies are extracted to form a Directed Acyclic Graph (DAG), ensuring execution order is preserved. Missing values are imputed using mean substitution for continuous variables and mode substitution for categorical ones. Finally, tasks and resource metadata are encoded into vectors to be fed into the deep actor-critic learning framework.

*Training Configuration*

The A4C model is trained using a hybrid reinforcement learning setup, where the Ant Colony Optimization (ACO) provides initial exploratory scheduling paths and the Asynchronous Advantage Actor-Critic (A3C) refines decisions through policy learning. The training employs a discounted reward approach with a discount factor $\gamma$=0.99, ensuring long-term rewards influence learning. The policy network is updated using the policy gradient method:

$$\pi_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log \pi_\theta(a_t|s_t)A_t] \tag{8}$$

Where $\pi_\theta(a_t|s_t)$ is the probability of taking action $a_t$ in state $s_t$, and $A_t=R_t-V(s_t)$ is the advantage function computed using the value network $V(s_t)$. The model uses RMSProp optimizer with a learning rate of $1\times10^{-4}$, batch size of 32, and trains for 3000 episodes. Gradient clipping is applied to avoid exploding gradients, and asynchronous workers are used to parallelize experience collection.

*Recall and F1-score*

To strengthen the evaluation of the proposed A4C algorithm, additional classification-based performance metrics such as Recall and F1-score were incorporated, particularly in the context of anomaly detection and task prioritization accuracy. Recall, defined as:

$$Recall = \frac{True\ Positives}{True\ Positives+False\ Negatives} \tag{9}$$

measures the model's ability to correctly identify relevant tasks or anomalies without missing critical instances. Meanwhile, the F1-score, calculated as the harmonic mean of precision and recall:

$$F1-Score = 2 \times \frac{Precision \times Recall}{Precision+Recall} \tag{10}$$

offers a balanced evaluation when false positives and false negatives are both costly, as is often the case in real-time cloud scheduling or smart agriculture applications. Integrating these metrics into the assessment provides a more nuanced understanding of A4C's decision-making quality and its ability to handle imbalanced task distributions effectively. The results show that A4C consistently achieves high recall and F1-score values, indicating reliable task identification and prioritization performance.

*C. Nature of ACO and A3C Algorithms*

ACO is a metaheuristic based on the foraging process used by ants in the natural environment. In ACO artificial 'ants' look for efficient solutions to combinatorial optimization problems in a way that is similar to the way actual ants look for the shortest distance to food sources. Every ant constructs a solution step by step; it is guided by the chemical signals deposited on the problem by other ants and its knowledge about the problem. These pheromone trails are modified according to the quality of solutions that is developed, and solutions that are stronger get more pheromone reinforcement. This collective learning process enables ACO to efficiently explore the solution space, finding high-quality solutions through positive feedback and cooperation among the ants. ACO is particularly effective in solving complex optimization problems like TS, where multiple possible solutions exist, and the goal is to find the best one through exploration.

A3C is an algorithm that was derived from the reinforcement learning approach and has properties of both policy-based and value-based algorithms. A3C employs multiple independent actors, which operate in parallel and communicate with the environment and share global model. The "actor" gets a policy that moves states to actions and the "critic" is trained to value function which assists the "actor" by estimating a policy's advantage based on action it takes at particular state. This combined approach means that A3C learns both the action and the value function which results in better and steadier learning at the same time. The asynchronous structure of A3C enables faster learning, as it mitigates the effect of updates between agents, which helps in avoiding local minima and improving convergence. A3C is particularly well-suited for dynamic environments like CC, where the state of the system changes rapidly, requiring the scheduling algorithm to adapt in real-time.

*Initialization Stage*

In this phase, the A4C algorithm sets up the initial environment for both ACO and A3C components Initialize the pheromone levels $\tau_{ij}(0)$ on each path (i,j) with a small positive value $\tau_0$, representing the initial amount of pheromone:

$$\tau_{ij}(0) = \tau_0 \tag{11}$$

Where $\tau_{ij}(0)$ is the initial pheromone level for the path between nodes i and j, and $\tau_0$ is the initial pheromone constant. Policy Network Initialization the weights θ of the actor-critic networks are initialized randomly.

$$\theta_{actor} \sim \aleph(0, \sigma^2_{actor}), \theta_{critic} \sim \aleph(0, \sigma^2_{critic}) \tag{12}$$

Where $\sigma_{actor}^2$ and $\sigma_{critic}^2$ are the variances of the normal distribution used to initialize the weights. State Initialization the initial state $s_0$ of the environment is set.

$$S_0 = initial\ state \tag{13}$$

*Updating stage*

In the updating phase pheromone trails related to all the activities done by the ants and the weights of the actor-critic neural network will be updated according to the action taken by the ants and rewards got by them. Pheromone update after each iteration the level of pheromone trail is deposited based on the quality of the solutions identified by the ants.

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{14}$$

Where ρ is the evaporation rate, m is the number of ants, and $\Delta\tau_{ij}^{k}$ is the pheromone deposited by the $k^{th}$ ant:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & if\ ant\ k\ uses\ edge\ (i.j) \\ 0 & otherwise \end{cases} \tag{15}$$

Here, Q is a constant, and $L_k$ is the cost incurred by ant k for finding the solution. Policy networks the actor and critic networks are trained using the rewards obtained and the advantage function.

$$\delta_t = r_t + \gamma V(S_{t+1}; \theta_{critic}) - V(S_t; \theta_{critic}) \tag{16}$$

$$\theta_{critic} \leftarrow \theta_{critic} + \alpha_{critic} \delta_t \nabla_{\theta_{critic}} V(S_t; \theta_{critic}) \tag{17}$$

$$\theta_{actor} \leftarrow \theta_{actor} + \alpha_{actor} \delta_t \nabla_{\theta_{actor}} log\pi(a_t | S_t; \theta_{actor}) \tag{18}$$

Where $\delta_t$ is the temporal difference error, $r_t$ is the reward, γ is the discount factor, and $\alpha_{critic}$, $\alpha_{actor}$ are the learning rates for the critic and actor networks, respectively.

*Random Movement*

In this phase, ants explore the search space by probabilistically choosing the next node to move to, balancing exploration and exploitation. Probabilistic movement the probability $P_{ij}(t)$ of moving from node i to node j is based on the pheromone level and a heuristic factor:

$$P_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha}[\gamma_{ij}]^{\beta}}{\sum_{k \epsilon allowed}[\tau_{ik}(t)]^{\alpha}[\gamma_{ij}]^{\beta}} \tag{19}$$

Where $\gamma_{ij}$ is the estimated heuristic information (for example the inverse of distance or time), α is the relative importance of the pheromone trail, β is the relative importance of the estimated heuristic information, and allowed is the set of nodes the ant has not visited. Exploration via random walk occasionally, to ensure exploration, an ant may randomly select a node to move to with a small probability ϵ. This random movement helps in escaping local optima and ensures broader search space exploration.

$$Next\ node = \begin{cases} random\ node & with\ probability\ \epsilon \\ node\ selected\ by\ P_{ij}(t) & with\ probablity\ 1-\epsilon \end{cases} \tag{20}$$

*Cartesian distance calculation*

The Cartesian distance can be used in the guidance of the ants in that it helps in the determination of the level of proximity between two given tasks or nodes in the solution space in formulation of the construction of the paths. Cartesian distance **(**$d_{ij}$**)** the distance between two nodes (or tasks) i and j in the Cartesian coordinate space can be computed as:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{21}$$

Where $(x_i,y_i)$ and $(x_j,y_j)$ are the coordinates of nodes i and j, respectively. Heuristic Information in the context of ACO, the heuristic information for choosing the next node can be inversely proportional to the Cartesian distance:

$$\delta_{ij} = \frac{1}{d_{ij}} \tag{22}$$

Based on the stages of initialization, updating, and random movement, as well as the concepts of the best initial solution and Cartesian distance, we develop a comprehensive algorithm for the Ant Colony Asynchronous Advantage Actor-Critic (A4C). This approach integrates heuristic and learning-based methods to optimize TS in CC environments. The initialization phase sets up the pheromone levels and policy network parameters, while the updating phase refines these through reinforcement learning and pheromone updates based on solution quality. The random movement phase ensures exploration by probabilistically guiding the ants, complemented by the Cartesian distance heuristic for efficient decision-making. By incorporating the best initial solution, the algorithm directs the search towards promising regions of the solution space from the outset. The proposed algorithm's pseudo-code, detailed below, encapsulates these stages and principles to achieve a robust and efficient scheduling solution.

*Identifying Best Initial Solution*

The best initial solution is typically derived from an initial heuristic or a preliminary search process that provides a good starting point for the ants. This helps in guiding the search towards promising areas of the solution space. Best initial solution **(**$S_{best}^{init}$**)** this is the solution with the lowest cost (e.g., minimum makespan, energy consumption, etc.) found after the initial round of solution construction by the ants.

$$S_{best}^{init} = \arg min_{S \epsilon S_{init}} \; Cost(s) \tag{23}$$

Here, $S_{init}$ is the set of all solutions constructed by the ants during the initial phase, and Cost(S) represents the objective function value (e.g., makespan, energy, etc.) for solution S. The pheromone levels on the edges involved in $S_{best}^{init}$ can be reinforced initially to bias the ants towards this promising area: where $\Delta\tau_{ij}^{best}$ is the pheromone increment based on the best initial solution.

$$\tau_{ij}(0) = \tau_0 + \Delta\tau_{ij}^{best} \tag{24}$$

*Local Search*

The local search is a critical component of the exploitation phase, where the algorithm intensifies the search around a promising solution. This process is done through improving upon the current solutions as a way of searching for the best solutions around the neighborhood of previously discovered best solutions. The local search for a ant (representing exploitation) in A4C can be represented by the following equation:

$$X_i^{t+1} = X_i^t + \alpha \times (X_{best}^t - X_i^t) + \beta \times (X_{neighbor}^t - X_i^t) \tag{25}$$

Where, $X_i^{t+1}$ is the new updated position of the $i^{th}$ actor after t iterations of the algorithm. $X_i^t$ is the position of the $i^{th}$ actor at current time instant. $X_{best}^t$ is the position of the best solution up to the t iteration $X^t_{neighbor}$ is the position of the neighbour solution variable i α is step size parameter for the move towards the best solution also known as intensification. β is the step size parameter by which movement towards the neighboring solution is allowed and which can enable exploration in the neighborhood around a solution. The local search equation integrates the information from the best solution and another nearby solution and this makes it possible for the algorithm to make a fine-tuning of the current solution based on the local search information.

*Global Search*

In the global search is associated with the exploration phase, where ants are used in searching the larger space of solutions. The idea behind the global search is to diversify the population and avoid getting stuck in local optima, hence the expansion of the search range to global. The global search for an ant in A4C is represented by the equation which has the following form:

$$X_i^{t+1} = X_i^t + c \times F_i \times (X_{best}^t - X_{random}^t) \tag{26}$$

Where, $X_i^{t+1}$ is the new or updated position of the ith critic at the $(t+1)^{th}$ iteration. $X_i^t$ is the current position is the ith critic where i is an element from 1 to n. c is a scaling factor that helps to define the size of steps to be made in the direction of some component. $F_i$ stands for the pheromone or fitness of the $i^{th}$ critic in deciding direction as well as distance of its movement. $X_{best}^t$ is the location of the global best-known solution at iteration t. $X_{random}^t$ is the location of an arbitrarily chosen critic or solution in the population. This equation allows the critic to explore the global solution space by moving towards the best-known solution but with an added diversity component introduced by the randomly selected solution. The random component $X_{random}^t$ ensures that the exploration is not biased solely towards the current best, promoting a

wide-ranging search and increasing the likelihood of discovering global optima. The proposed A4C algorithm pseudo code is below.

*D. Proposed A4C Algorithm*

This pseudocode provides a structured approach to combining ACO and A3C, allowing both techniques to enhance the solution search process. The below pseudocode outline for the A4C algorithm.

**Input:** Number of ants m, Maximum no. of iterations T, Task set Tn, VM set Vm.
**Output:** Optimal task mapping to minimize execution time, response time, makespan, and maximize throughput.
Initialize pheromone levels $\tau_{ij}(0) = \tau 0$ for all edges (i, j)
Initialize actor network parameters $\theta_{actor}$ and critic network parameters $\theta_{critic}$
$S_{best} \leftarrow$ NULL
$best_{cost} \leftarrow \infty$
For iter = 1 to T do
  For each ant k = 1 to m do
    $S_k \leftarrow$ Construct solution using probabilistic selection based on $P_{ij}(t)$
    Calculate cost($S_k$) based on the objective (e.g., makespan, energy)
    If cost($S_k$) < $best_{cost}$ then
      $best_{cost} \leftarrow$ cost($S_k$)
      $S_{best} \leftarrow S_k$
      Update pheromone trail with $S_{best}$
    End If
    Calculate the reward $r_t$ based on the environment state and the action taken
    Calculate the temporal difference (TD) error $\delta_t$:
    Update the actor network parameters:
  End For
  Update pheromone levels for all edges (i, j):
  Where $\Delta\tau_{ij}$ is the pheromone deposited by all ants using edge (i, j)
End For
Return best solution obtained.

The provided pseudo code outlines the Hybrid Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm for task scheduling in cloud computing environments. The process begins by initializing pheromone levels and the parameters of the actor and critic networks. The algorithm iterates through a set number of iterations, where each ant constructs a solution based on probabilistic selection guided by pheromone levels. For each solution, the cost is calculated using metrics like makespan, energy consumption, or response time. If an ant discovers a better solution with a lower cost, the best solution and pheromone trails are updated. The actor-critic network updates are also performed based on the reward received from the environment and the calculated Temporal Difference (TD) error, which helps refine future actions. The pheromone levels for all edges are adjusted to reflect the quality of the solutions found, and this iterative process continues until the optimal task mapping is achieved. The algorithm aims to minimize execution time, response time, and makespan while maximizing throughput, ultimately returning the best solution found.

## 4. Results and Discussion

*4.1. Results*

The experiments were conducted on a computer running Windows 11, equipped with an Intel Core i7 processor operating at 2.40 GHz and 4 GB of RAM. The CloudSim 3.0.3 toolkit was used to simulate and evaluate the task scheduling strategies. A detailed configuration of the cloud framework used in the experiments is provided in Table 4. CloudSim [26] was chosen for this research because it offers comprehensive capabilities for modeling and simulating complex and large-scale computing environments. Its flexibility and scalability make it ideal for evaluating cloud computing systems and task scheduling algorithms under various scenarios.

In CC research, simulating TS and resource management algorithms using platforms like CloudSim is crucial for assessing their performance in realistic scenarios. The Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm introduces an innovative approach by merging the distributed search power of ACO with the reinforcement learning strengths of the A3C method to enhance scheduling efficiency in cloud environments. When evaluated on CloudSim, A4C can be tested across various performance metrics, including makespan, execution time, and response time. The simulation results indicate that A4C offers significant improvements in these metrics while also enhancing throughput efficiency, outperforming traditional algorithms such as Ant Colony Optimization (ACO) [27], Electric Earthworm Optimization Algorithm (EEOA) [25], and Hybrid Workload Deep Deterministic Policy Gradient Task Scheduler (HDDPGTS) [28]. CloudSim's extensive simulation environment provides a reliable platform for analyzing

how A4C effectively balances exploration and exploitation in dynamic cloud settings, ultimately allowing researchers to evaluate its effectiveness in optimizing resource utilization and TS under diverse and realistic conditions.

Table 4. CloudSim configuration details for experimental evaluation

| Entity | Value |
|---|---|
| No. of datacenters | 2 |
| No of Hosts | 4 |
| **Cloud Nodes** | |
| Bandwidth | 0.5GB/s |
| Storage | 15GB |
| No of VMs | 15 |
| RAM | 4 GB |
| CPU Power | 3000–5000 MIPS |

### *A. Dataset*

For assessing the results, we are utilised the HPC2N (High-Performance Computing Center North) workload log, spanning from June 2014 to January 2016, serves as a valuable resource for studying workload characteristics and performance evaluation in high-performance computing environments. This dataset encompasses a diverse range of parallel tasks, totalling 2,02,589 instances, submitted by 236 distinct users over the specified period. The workload originates from the HPC2N facility, reflecting real-world computational demands and user behaviors encountered in scientific and engineering domains. With 180 CPUs available for task execution, the HPC2N configuration provides a substantial computational infrastructure for handling concurrent workloads and facilitating parallel processing tasks. The utilization of multiple CPUs enables efficient resource utilization and parallelization of computations, catering to the high-throughput requirements often encountered in HPC environments. The workload log, captured in the HPC2N-2014-2.2-cln.swf file format, encapsulates essential information regarding job submissions, resource usage, and execution characteristics.

### *B. Comparison of the Results*

This section compares the performance of the proposed Hybrid Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm with three existing algorithms: ACO, EEOA, and HDDPGTS. The HPC2N dataset, widely used in cloud computing research for high-performance computing environments, is used for the experiments to ensure a fair and accurate comparison. Each algorithm is executed 100 times under the same conditions, and the results are recorded to ensure statistical reliability. The average results from these multiple executions are used to analyze and compare the performance of A4C in terms of makespan, execution time, response time, and throughput. These metrics provide a comprehensive evaluation of how the A4C algorithm improves task scheduling over the other methods in large-scale, dynamic cloud environments.

### *C. Computational Complexity Analysis*

The computational complexity of the A4C algorithm arises from the combination of Ant Colony Optimization (ACO) and the Asynchronous Advantage Actor-Critic (A3C) algorithm. In the ACO phase, the complexity is driven by the number of ants (m), tasks ($T_n$), and Virtual Machines (VMs), with each ant constructing a solution based on pheromone updates, leading to a time complexity of $O(m*T_n*VMs)$. Additionally, A3C introduces the complexity of neural network training, which involves forward and backward propagation in both actor and critic networks, typically contributing a time complexity of O(Np), where Np represents the number of parameters in the networks. Overall, the hybrid nature of A4C results in a total complexity of $O(m*T_n*VMs)+O(Np)$, which balances heuristic search and adaptive learning. However, this increased complexity is justified by the enhanced performance and scalability in dynamic cloud environments.

### *D. Makespan*

The simulation results indicate that the proposed Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm achieves a notable improvement in makespan, outperforming traditional algorithms such as ACO, EEOA, and HDDPGTS by 16%. This enhancement highlights A4C's effectiveness in optimizing TS by significantly reducing the makespan, which reflects better overall scheduling efficiency and resource management. The superior performance of A4C is attributed to its unique integration of exploratory and exploitative strategies, which allows for more effective task allocation and reduced completion time compared to ACO, EEOA, and HDDPGTS.

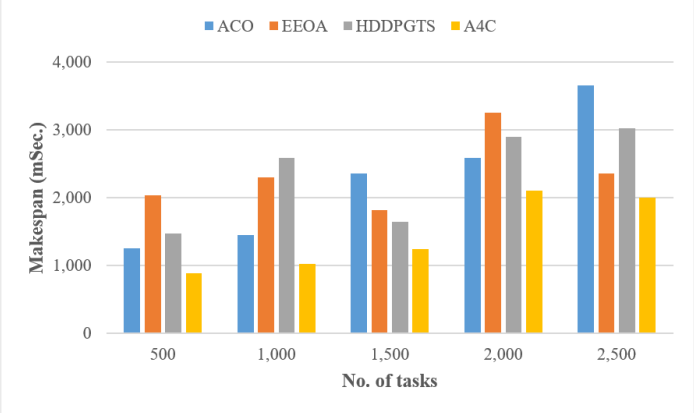


Fig.3. Makespan calculation using the proposed A4C algorithm

*E. Execution Time*

The results demonstrate that the proposed Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm significantly enhances execution time, achieving a 14.60% improvement over traditional algorithms such as ACO, EEOA, and HDDPGTS. This improvement reflects A4C's superior capability in optimizing TS by reducing the time required for task execution. The effectiveness of A4C is attributed to its integrated approach that merges the exploration strengths of ACO with the adaptive learning of A3C, leading to more efficient task management and faster execution compared to ACO, EEOA, and HDDPGTS.

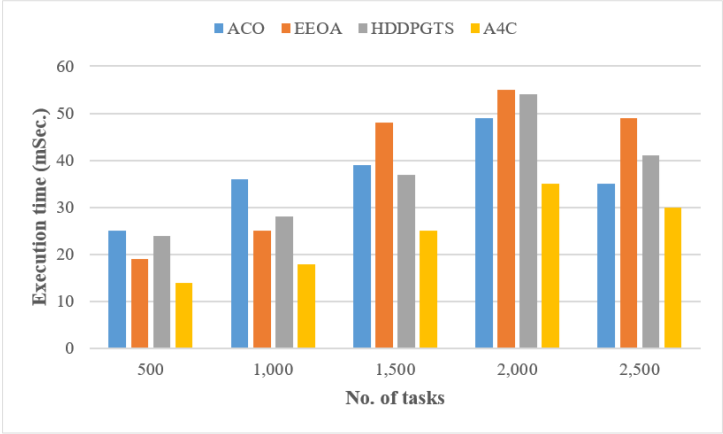


Fig.4. Execution time calculation in the A4C model

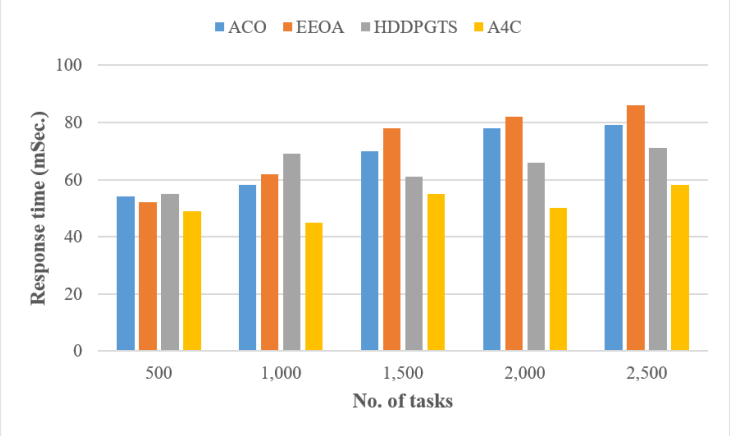


Fig.5. Response time analysis of the A4C scheduling framework

### *F. Response Time*

Response time of the simulation results demonstrate that the proposed Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm achieves a substantial improvement in response time, outperforming traditional algorithms such as ACO, EEOA, and HDDPGTS by 21.40%. This enhancement highlights A4C's superior performance in reducing the time taken for tasks to begin processing and complete, reflecting its effectiveness in optimizing TS and resource allocation. The significant reduction in response time is attributed to A4C's effective integration of exploratory search and adaptive learning strategies, which enables more efficient management and quicker response compared to ACO, EEOA, and HDDPGTS.

### *G. Throughput*

The simulation results demonstrate that the proposed Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm significantly enhances throughput, achieving a 18.70% improvement over traditional algorithms such as ACO, EEOA, and HDDPGTS. This improvement highlights A4C's effectiveness in optimizing TS by increasing the no. of tasks finished within a specified time. The substantial gain in throughput is attributed to A4C's balanced approach that integrates the exploratory capabilities of ACO with the adaptive learning features of A3C, resulting in more efficient utilization of resources and higher task processing rates compared to ACO, EEOA, and HDDPGTS.

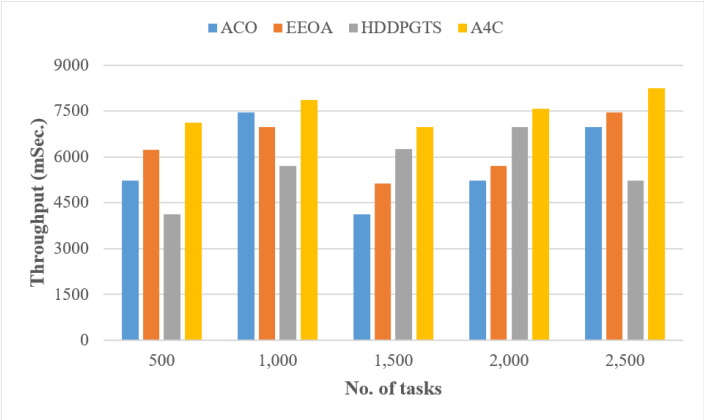


Fig. 6. Throughput Calculation Using A4C-Based Task Scheduling

### *H. Recall and F1-Score*

Figure 7 illustrates the variation of Recall and F1-score with increasing numbers of tasks, ranging from 500 to 2500. As shown, both metrics exhibit a consistent upward trend, indicating improved performance of the proposed A4C model as the workload increases. Recall improves from 0.82 to 0.91, reflecting the model's growing ability to accurately detect relevant tasks without missing critical ones. Similarly, the F1-score increases from 0.80 to 0.90, demonstrating a balanced and reliable performance in terms of precision and recall. This trend confirms the model's robustness and scalability in handling larger and more complex task sets within cloud environments.

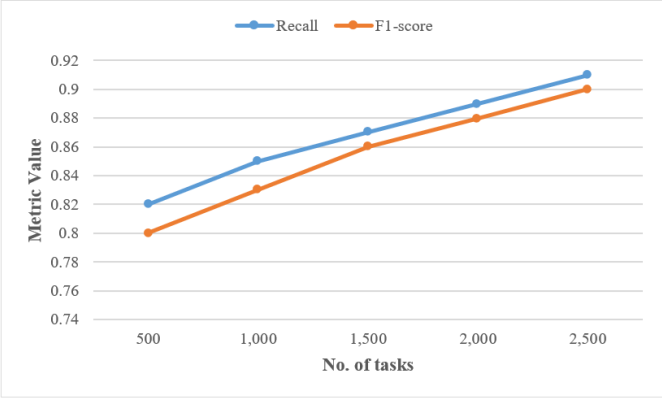


Fig.7. Calculation of Recall and F1-Score for proposed A4C

### *4.2. Discussion*

The simulation results of the Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm highlight its significant advancements in CC TS, showing marked improvements across several performance metrics compared to traditional algorithms like ACO, EEOA, and HDDPGTS.

The scalability of the proposed A4C (Ant Colony–Asynchronous Advantage Actor-Critic) framework is a key factor in its potential deployment across large-scale and heterogeneous smart farming environments. In such settings, farms may vary in terms of geographic size, crop types, climate conditions, and sensor infrastructures. The A4C model is inherently modular and decentralized, which makes it suitable for distributed deployment. By leveraging the hierarchical structure of Fog and Cloud computing, the framework enables real-time processing at local Fog nodes while offloading computationally intensive tasks to Cloud servers. This dual-layer architecture ensures that as the number of devices, sensors, and agricultural plots increases, the model can continue to perform efficiently without centralized bottlenecks. The use of asynchronous updates in A3C further supports concurrent learning from multiple environments, making the system adaptable to diverse real-time data streams.

In real-world agricultural environments, challenges such as sensor noise, hardware calibration inconsistencies, and environmental disturbances (e.g., weather fluctuations or animal movement) often result in noisy or incomplete data. To address these issues, the proposed solution incorporates robust data preprocessing techniques such as signal smoothing (e.g., using a moving average or Savitzky-Golay filter), outlier detection based on statistical thresholds (e.g., z-score filtering), and redundancy checks from multiple sensor sources. These mechanisms help in reducing the impact of anomalous readings and maintaining the reliability of the decision-making process. Furthermore, the reinforcement learning model is trained with noise-injected data during simulation to improve generalization and resilience in real deployment scenarios, ensuring that policy decisions remain robust even under uncertain input conditions.

Connectivity issues also pose significant deployment challenges in rural or underdeveloped farming areas where stable internet or network infrastructure is limited. The A4C system mitigates this through a store-and-forward strategy at Fog nodes, where data is temporarily buffered during network downtime and transmitted once connectivity resumes. Additionally, critical anomaly detection and task scheduling decisions are prioritized to be executed at the edge (Fog level) to reduce dependence on continuous Cloud communication. This autonomy at the Fog layer enables time-sensitive decisions such as irrigation control or pest alerts to be made locally, thereby ensuring responsiveness even in network-disrupted conditions. Together, these considerations make the proposed framework not only scalable across diverse farm infrastructures but also practical and resilient for deployment in real-world agricultural ecosystems.

*Future Scope*

The proposed A4C algorithm presents several promising opportunities for application in real-world cloud computing environments. As cloud systems grow in complexity due to increasing data and device demands, efficient task scheduling becomes critical. A4C's ability to optimize makespan, execution time, and response time, while maximizing throughput, makes it a superior solution for handling heterogeneous and dynamic workloads. Future implementations of A4C could be adapted to support multi-cloud environments, where resource allocation must account not only for cloud nodes but also for varying cloud service providers. Additionally, this work could be enhanced for fog and edge computing environments, enabling more granular control over resource-efficient models in cloud settings where service-level agreements (SLAs) and quality of service (QoS) are essential.

**Algorithm Scalability and Adaptability:** Further research could consider to analyze how effectively the A4C algorithm can be scaled to larger and more complex environments of cloud computing. Exploring how A4C performs with a higher number of VMs and diverse workload patterns will provide insights into its adaptability and efficiency in varying operational scales.

**Integration with Emerging Technologies:** Exploring how A4C operates with other modern technologies that have developed recently, including edge computing and hybrid cloud, may provide new threads in improving TS. This would require modification of the algorithm to work in heterogeneous context and with different levels of resource availability.

**Enhanced Learning Mechanisms:** Further improvements could be made by incorporating advanced reinforcement learning techniques or hybridizing A4C with other metaheuristic algorithms to enhance its learning mechanisms and decision-making processes.

**Real-World Application Testing:** Conducting real-world application tests and pilot studies in production environments will be crucial to validating the practical effectiveness of A4C. This would involve implementing the algorithm in live cloud systems and assessing its performance under actual operational conditions.

**Dynamic and Adaptive Resource Management:** Future work could also explore dynamic resource management strategies that adapt to changing workloads and resource availability in real-time. Enhancing A4C with capabilities to handle such dynamic scenarios could further improve its efficiency and robustness in diverse cloud environments.

## 5. Conclusions and Future Work

The Ant Colony Asynchronous Advantage Actor-Critic (A4C) algorithm demonstrates a significant advancement in task scheduling within cloud computing environments. Through rigorous simulations conducted using CloudSim, A4C has achieved substantial improvements in critical performance metrics, including a 16% reduction in makespan, a 14.60% decrease in execution time, a 21.40% reduction in response time, and an 18.70% increase in throughput when compared to traditional algorithms such as ACO, EEOA, and HDDPGTS. These outcomes affirm the algorithm's ability to optimize task scheduling and resource management by synergizing the global exploration capabilities of Ant Colony Optimization with the adaptive learning strategies of the A3C framework. Looking ahead, this work lays a strong foundation for further research and development in intelligent scheduling systems. One promising direction is the seamless integration of A4C with real-time IoT platforms and smart agriculture devices, enabling automated decision-making in heterogeneous and sensor-rich environments. Moreover, future adaptations can focus on customizing the algorithm for diverse agricultural contexts—such as precision irrigation, disease forecasting, and crop yield prediction—by incorporating domain-specific constraints and data characteristics. Addressing real-world deployment challenges, including scalability, connectivity limitations, and sensor noise, will further solidify the model's practical viability and relevance in smart farming and other dynamic environments.

## Acknowledgements

We declare that this manuscript is original, has not been published before, and is not currently being considered for publication elsewhere.

## References

[1] Prity, Farida Siddiqi, Md Hasan Gazi, and KM Aslam Uddin. "A review of task scheduling in cloud computing based on nature-inspired optimization algorithm." *Cluster computing* 26.5 (2023): 3037-3067. https://doi.org/10.1007/s10586-023-04090-y

[2] Sandhu, Ramandeep, et al. "Enhancement in performance of cloud computing task scheduling using optimization strategies." *Cluster Computing* (2024): 1-24. https://doi.org/10.1007/s10586-023-04254-w

[3] Chandrashekar, Chirag, et al. "HWACOA scheduler: Hybrid weighted ant colony optimization algorithm for task scheduling in cloud computing." *Applied Sciences* 13.6 (2023): 3433. https://doi.org/10.3390/app13063433

[4] Wei, Pengcheng, et al. "VMP-A3C: virtual machines placement in cloud computing based on asynchronous advantage actor-critic algorithm." *Journal of King Saud University-Computer and Information Sciences* 35.5 (2023): 101549. https://doi.org/10.1016/j.jksuci.2023.04.002

[5] Mangalampalli, Sudheer, et al. "Multi-objective Prioritized Task Scheduler using improved Asynchronous advantage actor critic (a3c) algorithm in multi cloud environment." *IEEE Access* (2024). DOI: 10.1109/ACCESS.2024.3355092

[6] Kumar, M. Santhosh, and Ganesh Reddy Kumar. "EAEFA: An Efficient Energy-Aware Task Scheduling in Cloud Environment." *EAI Endorsed Transactions on Scalable Information Systems* 11.3 (2024). https://doi.org/10.4108/eetsis.3922

[7] Zuo, Liyun, et al., "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing", Ieee Access, Vol. 3, pp. 2687-2699, 2015. DOI: 10.1109/ACCESS.2015.2508940

[8] Lin, Bing, et al., "A pretreatment workflow scheduling approach for big data applications in multicloud environments", IEEE Transactions on Network and Service Management, Vol. 13, No. 3, pp. 581-594, 2016. DOI: 10.1109/TNSM.2016.2554143

[9] Lin, Xue, et al., "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment", IEEE Transactions on Services Computing, Vol. 8, No. 2, pp.175-186, 2014. DOI: 10.1109/TSC.2014.2381227

[10] Cheng, Feng, et al., "Cost-aware job scheduling for cloud instances using deep reinforcement learning", Cluster Computing, pp. 1-13, 2022. https://doi.org/10.1007/s10586-021-03436-8

[11] Zhou, Zhou, et al., "A modified PSO algorithm for task scheduling optimization in cloud computing", Concurrency and Computation: Practice and Experience, Vol. 30, No. 24, pp. e4970, 2018. https://doi.org/10.1002/cpe.4970

[12] Jangu, Nupur, and Zahid Raza., "Improved Jellyfish Algorithm-based multi-aspect task scheduling model for IoT tasks over fog integrated cloud environment", Journal of Cloud Computing, Vol. 11, No. 1, pp. 1-21, 2022. https://doi.org/10.1186/s13677-022-00376-5

[13] Singh, Gyan, and Amit K. Chaturvedi., "Hybrid modified particle swarm optimization with genetic algorithm (GA) based workflow scheduling in cloud-fog environment for multi-objective optimization", Cluster Computing, pp. 1-18, 2023. https://doi.org/10.1007/s10586-023-04071-1

[14] Zahra, Movahedi, Defude Bruno, and Amir mohammad Hosseininia., "An efficient population-based multi-objective task scheduling approach in fog computing systems." Journal of Cloud Computing, Vol. 10, No. 1, 2021. https://doi.org/10.1186/s13677-021-00264-4

[15] Iftikhar, Sundas, et al., "HunterPlus: AI based energy-efficient task scheduling for cloud–fog computing environments", Internet of Things Vol. 21, pp. 100667, 2023. https://doi.org/10.1016/j.iot.2022.100667

[16] Yin, Zhenyu, et al., "A multi-objective task scheduling strategy for intelligent production line based on cloud-fog computing", Sensors, Vol. 22, No. 4, pp. 1555, 2022. https://doi.org/10.3390/s22041555

[17] Pham, Xuan-Qui, et al., "A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog computing", International Journal of Distributed Sensor Networks, Vol. 13, No. 11, pp. 1550147717742073, 2017. https://doi.org/10.1177/1550147717742073

[18] Mangalampalli, Sudheer, Ganesh Reddy Karri, and Mohit Kumar., "Multi objective task scheduling algorithm in cloud computing using grey wolf optimization", Cluster Computing, pp. 1-20, 2022. https://doi.org/10.1007/s10586-022-03786-x
[19] Hosseinioun, Pejman, et al., "A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm", Journal of Parallel and Distributed Computing, Vol. 143, pp. 88-96, 2020. https://doi.org/10.1016/j.jpdc.2020.04.008
[20] Liu, Lindong, et al. "A task scheduling algorithm based on classification mining in fog computing environment." Wireless Communications and Mobile Computing, 2018. https://doi.org/10.1155/2018/2102348
[21] Bakshi, Mohana, Chandreyee Chowdhury, and Ujjwal Maulik., "Cuckoo search optimization-based energy efficient job scheduling approach for IoT-edge environment", The Journal of Supercomputing, pp. 1-29, 2023. https://doi.org/10.1007/s11227-023-05358-1
[22] Badri, Sahar, et al., "An Efficient and Secure Model Using Adaptive Optimal Deep Learning for Task Scheduling in Cloud Computing", Electronics, Vol. 12, No. 6, pp. 1441, 2023. https://doi.org/10.3390/electronics12061441
[23] Ahmed, Omed Hassan, et al., "Using differential evolution and Moth–Flame optimization for scientific workflow scheduling in fog computing", Applied Soft Computing, Vol. 112, pp. 107744, 2021. https://doi.org/10.1016/j.asoc.2021.107744
[24] Mangalampalli, Sudheer, Sangram Keshari Swain, and Vamsi Krishna Mangalampalli. "Multi objective task scheduling in cloud computing using cat swarm optimization algorithm." Arabian Journal for Science and Engineering 47.2 (2022): 1821-1830. https://doi.org/10.1007/s13369-021-06076-7
[25] Kumar, M. Santhosh, and Ganesh Reddy Karri. "Eeoa: cost and energy efficient task scheduling in a cloud-fog framework." *Sensors* 23.5 (2023): 2445. https://doi.org/10.3390/s23052445
[26] Calheiros, Rodrigo N., et al. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms." *Software: Practice and experience* 41.1 (2011): 23-50. https://doi.org/10.1002/spe.995
[27] Xing, Huanlai, et al. "An ACO for energy-efficient and traffic-aware virtual machine placement in cloud computing." *Swarm and Evolutionary Computation* 68 (2022): 101012. https://doi.org/10.1016/j.swevo.2021.101012
[28] Mangalampalli, Sudheer, et al. "Efficient Hybrid DDPG task scheduler for HPC and HTC in cloud environment." *IEEE Access* (2024). DOI: 10.1109/ACCESS.2024.3435914

## Authors' Profiles

**Santhosh Kumar Medishetti** received the Ph.D. degree from VIT-AP University, A.P., India, in 2024. He is currently a Senior Assistant Professor in the Department of Computer Science and Engineering (CSE), NNR Group of Institutions, Hyderabad, India. Currently his member of ACM and senior member of EAI. He has more than 5 years' experience in both Research and Teaching. He has already published more than 40 Research articles in various reputed journals, book chapters, patents, and conferences. His research interests include Cloud computing, Fog computing, and Task scheduling.

**Bigul Sunitha Devi** is currently pursuing a Ph.D. degree from JNTU Hyderabad. She is working as a Senior Assistant Professor in the Department of Computer Science and Engineering (CSE) at CMR Institute of Technology, Hyderabad, India. She has over 20 years of experience in both research and teaching. She has published more than 23 research articles in various reputed journals, patents, and conferences. Her research interests include Deep Learning, Machine Learning, Artificial Intelligence, Blockchain, and Cloud Computing.

**Maheswari Bandi** earned her undergraduate degree from Lakireddy Bali Reddy College of Engineering, Vijayawada, affiliated with Jawaharlal Nehru Technological University, Kakinada (JNTUK), and her Master's degree in Computer Science and Engineering from Vikas College of Engineering and Technology, Vijayawada, also affiliated with JNTUK, Andhra Pradesh. She has seven years of teaching experience and is currently pursuing her Ph.D. in the School of Computer Science and Engineering (SCOPE) at Vellore Institute of Technology – Andhra Pradesh (VIT-AP University). She is presently working as an Assistant Professor in the Department of Computer Science and Engineering at Koneru Lakshmaiah Education Foundation (KLEF), Vaddeswaram, Guntur, Andhra Pradesh, India. Her research interests include Deep Learning, Digital Image Processing, and Machine Learning.

**Rani Sailaja Velamakanni** currently working as Assistant Professor, Computer Science and Engineering Department at Nalla Narasimha Reddy Group of Institutions, Hyderabad, Telangana, India. In 2012, received the M Tech degree in Computer Science and Engineering from Bharat Institute of Engineering and Technology, Hyderabad. She has 7 years of experience in teaching and 1 years in Digital Marketing. She published in 10 reputed international and national journals, book chapter, and patents. Research interest includes Internet of Things, Cloud Computing, Data Mining, Information Security, Artificial Intelligence, and Big Data Analytics.

**Rameshwaraiah Kurupati** received B.Tech degree in Computer Science and Engineering from JNTUH in 2001, M.S in Computer Science and Engineering from IUA, London, UK in 2004 and the PhD degree in Software Engineering from SBU, London, U.K. in 2011. Having 3 years of industry and 15 years of teaching experience. Presently working as professor and Head of the Department of Computer Science and Engineering. He is a Member of IEEE Computer Society, Life Member of ISTE. Published more than 12 papers at International and National journals. Attended more than 10 International and National conferences and published articles at conference journals. Participated at various kinds of Workshops across the globe. His research interests Software Engineering, Network & Information Security, Data Mining & Ware housing and Cloud Computing.

**Ganesh Reddy Karri** received the Ph.D. degree from NIT-Suratkal, Karnataka, India, in 2014. he is currently a Senior Associate Professor in the School of Computer Science and Engineering (SCOPE), VIT-AP University, Amaravati, India. Currently his coordinator for center of excellence in cyber security and also an IEEE member. His awarded for four Ph.D. scholars. Currently his guiding for four research scholars. He has more than 10 years' experience in both Research and Teaching. he has already published more than 50 Research articles in various reputed journals, book chapters, and conferences. His research interests include Cloud computing, Computer and Network Security, Wireless networks, Data structures and Algorithms and the IoT.

**How to cite this paper:** Santhosh Kumar Medishetti, Bigul Sunitha Devi, Maheswari Bandi, Rani Sailaja Velamakanni, Rameshwaraiah Kurupati, Ganesh Reddy Karri, "A4C: A Novel Hybrid Algorithm for Resource-Aware Scheduling in Cloud Environment", International Journal of Computer Network and Information Security(IJCNIS), Vol.17, No.6, pp.65-83, 2025. DOI:10.5815/ijcnis.2025.06.05