Swarm and Evolutionary Computation 92 (2025) 101756

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Regular Paper

Integrating asynchronous advantage actor–critic (A3C) and coalitional game theory algorithms for optimizing energy, carbon emissions, and reliability of scientific workflows in cloud data centers

Mustafa Ibrahim Khaleel

Department of Computer, College of Science, University of Sulaimani, Sulaymaniyah, 46001, Iraq

ARTICLE INFO

Keywords:
Coalitional game theory
Carbon emissions
A3C RL
Energy consumption
Scheduling reliability

ABSTRACT

The growth of workflow as a service (WFaaS) has become more intricate with the increasing variety and number of workflow module applications and expanding computing resources. This complexity leads to higher energy consumption in data centers, negatively impacting the environment and extending processing times. Striking a balance between reducing energy and carbon emissions and maintaining scheduling reliability is challenging. While deep reinforcement learning (DRL) approaches have shown significant success in workflow scheduling, they require extensive training time and data due to application homogeneity and sparse rewards, and they do not always guarantee effective convergence. On the other hand, experts have developed various scheduling policies that perform well for different optimization goals, but these heuristic strategies lack adaptability to environmental changes and specific workflow optimization. To address these challenges, an enhanced asynchronous advantage actor–critic (A3C) method combined with merge-and-split-based coalitional game theory is proposed. This approach effectively guides DRL learning in large-scale dynamic scheduling issues using optimal policies from the expert pool. The merge-and-split-based method prioritizes computing nodes based on their preemptive characteristics and resource heterogeneity, ensuring reliability-aware workflow scheduling that maps applications to computing resources while considering the dynamic nature of energy costs and carbon footprints. Experiments on real and synthesized workflows show that the proposed algorithm can learn high-quality scheduling policies for various workflows and optimization objectives, achieving energy efficiency improvements of 7.65% to 19.32%, carbon emission reductions of 3.13% to 14.76%, and reliability enhancements of 17.22% to 41.65%.

1. Introduction

The cloud scheduler (also known as the broker) in cloud systems plays a crucial role in optimally distributing workflows across a limited number of virtual machines (VMs). This function is essential for mediating between users and providers, managing users' increasing demands for resource availability in data centers, and balancing competing objectives such as energy efficiency, CO2 emissions reduction, reliability, and fault-free operation [1]. Workflow scheduling is key to the performance of cloud systems and dramatically affects their efficiency [2]. However, it is recognized as an NP-complete problem, underscoring the challenge of finding the best solution within a reasonable timeframe [3]. Given the vast amounts of data that users and data centers manage in cloud computing, numerous studies have tackled workflow scheduling challenges. While many of these studies aim to optimize a single objective within specific constraints, high-performance computing systems often must address multiple, potentially conflicting optimization goals.

It has become clear that integrating game theory, such as coalitional-based cooperative game theory, with reinforcement learning techniques, like an improved version of asynchronous advantage actor–critic (A3C) that addresses multiple objectives, can effectively find near-optimal solutions for workflow scheduling. These hybrid algorithms handle multi-objective optimization problems well, offering various solutions that balance conflicting objectives, allowing the cloud scheduler to choose the best option based on computing resources' dynamic nature and preemptive characteristics. However, as explored in the literature, many existing reinforcement learning techniques for workflow scheduling are tailored to specific problems and are not directly applicable to our context. These algorithms are usually designed for single-objective tasks, making them inadequate for multi-objective optimization in workflow scheduling since they produce only

E-mail address: mustafa.khaleel@univsul.edu.iq.

https://doi.org/10.1016/j.swevo.2024.101756
Received 3 July 2024; Received in revised form 29 August 2024; Accepted 10 October 2024
Available online 29 November 2024

one solution and require scalarization to convert multiple objectives into a single one [4]. Therefore, our research adopts a collaborative-based coalitional cooperative game theory with an improved variation of A3C reinforcement learning for scheduling scientific workflows. The goal is to improve energy efficiency, ensure reliability (including server and communication link stability), and reduce CO2 emissions.

1.1. Research motivation

The COVID-19 pandemic and the rise of modern computing have driven a significant shift to cloud-based operations [5], dramatically increasing energy consumption by data centers. To illustrate, a single data center can use as much electricity as 25,000 households [6]. In 2020, global data centers consumed an astounding 73 billion kWh of electricity [7]. This trend is expected to continue, with projections indicating a 66% increase in electricity costs from 2011 to 2035 [8,9]. By 2030, the IT sector could use up to 13% of the world's electricity, with an annual consumption growth rate of 12% [10]. These statistics highlight the urgent need for energy-efficient practices in cloud computing [11]. The rise in data center energy consumption presents not only financial challenges but also significant environmental concerns. The Environmental Protection Agency (EPA) estimates that every 1000 kWh of power used generates approximately 0.72 metric tons of CO2 emissions [12], significantly contributing to global carbon dioxide emissions.

Inadequate workflow scheduling and uneven workflow allocations exacerbate the problem. For instance, over six months, about 5000 servers operated at only 10% to 50% of their capacity, leading to energy wastage [13]. Idle machines still consume a substantial amount of energy – approximately 60% to 80% of their total capacity energy consumption – regardless of workload. These inefficiencies underscore the necessity for more sustainable practices in cloud computing.

While energy consumption is a significant challenge, cloud computing also faces issues like server and communication failures, bandwidth utilization, asset utilization, and service quality, all of which affect the performance and reliability of cloud infrastructures. As the IT sector's energy consumption continues to rise, cloud service providers and users must adopt energy-efficient practices to mitigate environmental impact and ensure the long-term sustainability of cloud computing.

1.2. Research significance

The continual increase in data generated from various heterogeneous data centers complicates energy and carbon awareness and challenges server and communication failure rates. An effective strategy for managing applications in geo-distributed data centers involves designing a multi-level, multi-objective decision-making paradigm. However, offloading applications to this paradigm presents research challenges due to the need to consider multiple constraints before workflow scheduling. These constraints include the heterogeneity of computing machines, the dynamic nature of workflow applications, and server and communication failure rates.

To address these challenges, we classify workflow applications based on their preemptive characteristics. Applications with critical deadlines are assigned to powerful computing machines within a data center, while other applications are processed using low-power computing machines. We employ a merge-and-split-based coalitional game theory technique to cluster processing nodes and identify those with a high efficient-to-utilization ratio (EUR).

Additionally, we integrate the asynchronous advantage actor–Critic (A3C) method into coalitional game theory (CGT) to form a paradigm called (A3C-CGT) to optimize resource allocation decisions in geo-distributed data centers. This integration aims to enhance the efficiency and reliability of resource utilization while minimizing energy consumption and carbon emissions.

1.3. Research contribution

Managing scientific workflows involves focusing on two main areas. Firstly, allocating VMs based on the workflow's dynamic nature and needs is crucial. Over-provisioning resources can lead to unnecessary server activations, which reduces system efficiency and effectiveness. Evenly distributing workflow applications across active s prevents any VM from becoming overloaded, cutting energy costs and carbon footprints. Secondly, consolidating functional VM groups onto fewer operational servers is essential to lowering the risk of failure. Continuous monitoring of VM performance is necessary to spot potential issues, such as failures in communication links.

Predicting future requests accurately is vital for efficient resource allocation adjustments. Incorporating this predictive model helps prevent VM overloads and enhances the overall quality of service (QoS). Unlike previous studies, our research tackles multiple optimization objectives from workflow and resource perspectives, presenting a more practical and complex challenge. Therefore, considering various objectives, our study introduces a custom reliability-aware scientific workflow enhanced with A3C and CGT approaches. This algorithm is tailored to the specific problem and includes effective mechanisms to balance diversity and convergence, increasing efficiency. The main contributions of this paper are outlined as follows:

- A novel approach is introduced, integrating coalitional game theory with an enhanced version of the asynchronous advantage actor–critic (A3C) algorithm. This integration enables optimal decision-making for destinations in both local and global contexts.
- The operational servers are meticulously trained under different utilization loads using merge-and-split-based coalitional game theory to achieve the highest EUR. This approach forms a stable server group with minimal failure rates, ensuring the system's stability.
- Utilizing an improved asynchronous advantage actor–critic (A3C) algorithm, our model further classifies server usage, concentrating on patterns that maximize the EUR. A detection algorithm identifies and migrates overcrowded and undercrowded VMs to enhance resource utilization.
- The real and synthesized datasets have been dynamically managed and divided into quartiles using an interquartile range approach. This adaptive segmentation allows real-time threshold adjustments, providing a secure and responsive computing environment.

1.4. Research scope

The research discussed in this paper is closely aligned with the primary themes of "Swarm and Evolutionary Computation". This journal emphasizes tackling challenging search and optimization problems, including multi-objective, multimodal, dynamic, large-scale optimization, and algorithm hybridization. Our work contributes to these areas by examining the integration of asynchronous advantage actor–critic (A3C) with coalitional game theory, incorporating merge and split-based strategies. "Swarm and Evolutionary Computation" focuses on the theoretical, experimental, and practical dimensions of effective paradigms and their hybridizations, and our study aligns with these goals by advancing theoretical knowledge and providing practical insights for optimizing complex systems.

This study explores the collective behaviors and adaptive capabilities of the A3C-CGT approach in addressing complex optimization challenges, which is a key interest of the journal. By integrating reinforcement learning, we aim to enhance our understanding of how A3C-CGT can be utilized for dynamic decision-making and optimization in multi-agent systems. Additionally, our research demonstrates the potential of evolutionary algorithms to refine learning processes within A3C frameworks. Our study's game theory integration with A3C reinforcement learning is designed to tackle specific issues in multi-agent systems and distributed optimization. By employing game-theoretic models, we

offer a strategic framework that improves decision-making processes within swarms and evolutionary algorithms, thereby broadening the scope of these methodologies.

The remainder of the article is structured as follows: Section 2 covers previous studies. Section 3 presents the system design. Section 4 includes the mathematical models. Section 5 formulates the task scheduling problem. Section 6 details the algorithm. In Section 7, the proposed A3C-CGT is compared with MSISCSOA [14], SDAR-NFAT [15], DCOHHOTS [16], A2C [17], DQN [18], and EBABC-PF [19], and the results are discussed. Section 8 provides statistical analysis. Section 9 evaluates the proposed A3C-CGT. Finally, Section 10 concludes the article and suggests future directions.

2. Previous studies

This section concisely overviews previous research, concentrating on workflow scheduling methods in cloud computing systems.

In [20], a new optimization technique aims to reduce tardiness costs and power expenditures by introducing a hybrid PSO method. This method features direct updates of job operations and machine assignments in the discrete domain, a multi-objective tabu search approach, and a position-based crossover operator to balance global exploration and local exploitation. Our approach differs by using an approximate procedure to determine the optimized preferred utilization range for all computing servers, facilitating the efficient identification of optimal utilization levels and enabling the prediction of the system's highest efficiency-to-utilization ratio.

In [21], the Improved Beetle Swarm Optimization method is proposed for energy-conscious VM consolidation. This algorithm merges elements from Beetle Swarm Optimization and PSO techniques, optimizing power consumption and resource utilization and ensuring compliance with service level agreements (SLAs). Our algorithm stands out because it considers factors like network link failure rate and dynamic resource utilization, mitigating the risk of network bottlenecks.

In [22], a method for optimizing energy consumption in VM consolidation is presented. Their novel algorithm and energy consumption model significantly reduce energy consumption, achieving a 42% average reduction. Our approach diverges as they overlook parameters such as dynamic task changes, different VM types, and overhead from adjacent VMs.

In [23], the VM mapping problem is addressed to reduce power consumption costs for processing servers and network switches. The proposed "energy and traffic conscious, ETA-ACO" algorithm includes strategies like an energy and bandwidth optimization node selection method, a traffic VM ordering technique, and a direct information exchange approach. Our approach differs by considering scheduling reliability, memory-intensive, input/output-intensive tasks, and migration costs, which they overlook.

In [24], the PACS approach is proposed for dispatching containers to VMs to minimize energy consumption, costs, and request processing time. The PACS algorithm arranges VMs into groups for iterative allocation to cloudlets. Still, it does not consider preemptive VM migration, which leads to hosting numerous applications on a single computing node. Our approach addresses this gap.

In [25], the Massive Data Similarity Statistics Analysis Algorithm (MSSA) is presented, using statistical analysis to assess similarities between input workflows and categorize executing resources into virtual groups. MSSA also uses load balancing through resource exchange among VMs. Our paradigm distinguishes itself by considering fault tolerance mechanisms and enhancing task scheduling efficiency.

In [26], the Multi-Objective Hybrid Genetic Paradigm is proposed to address a scheduling problem involving two agents and energy constraints. This method integrates PSO to prevent premature convergence. Our algorithm differs by considering a window matrix to set VM time slots, reducing idle overhead between adjacent VMs.

In [27], the Chaotic-nondominated-sorting Owl Search model is proposed for scheduling workflows in hybrid clouds with resource limitations. Their method combines the Owl Search Algorithm (OSA) with a Nondominated Sorting Genetic Algorithm II (NSGA-II) to minimize energy-efficient deadlines and operational expenses. Our model includes additional factors like workload, host type, and VM overhead, enhancing system performance.

In [28], the Deadline-constrained Energy-aware Workflow Scheduling (DEWS) algorithm optimizes workflow scheduling with specific deadlines, considering energy consumption. DEWS arranges workflow tasks, searches for appropriate data centers, adjusts task sequences, and searches for suitable VMs with DVFS. Our approach considers the preferred CPU utilization range, which DEWS overlooks.

In [29], the Critical Parent Reliability-based Scheduling method prioritizes reliability in dispatching applications, considering user demands and deadlines. However, it fails to investigate the energy function thoroughly, which our approach addresses to enhance system effectiveness.

In [30], a multi-objective optimization approach combining HEFT and BAT algorithms tackles energy consumption, cost, makespan, and resource utilization. Our method offers extended computational complexity and a comprehensive energy function analysis, enhancing system efficiency.

In [19], a combined strategy involving BABC and PDS is introduced for scheduling workflow module applications, focusing on QoS requirements. Our approach distinguishes itself by implementing adaptable thresholds based on computing environment characteristics and considering the type of application running in VMs, reducing operational costs.

In [18], a cost-efficient application scheduler is proposed to manage real-time workload scheduling across a set of VMs to optimize operational costs. A deep reinforcement learning approach, specifically DQN, is adopted to implement and adjust the expense model. Parameters are trained and evaluated using PyTorch.

In [17], an eco-friendly federated-based cloud system (ERLFC) is proposed, utilizing reinforcement learning for workload placement in a federated cloud network. This paradigm monitors data center states and optimizes energy and CO2 rates across multiple data centers. The actor–critic algorithm selects the appropriate data center for housing assigned workloads.

In summary, various approaches in the literature address and provide solutions for task scheduling. However, a deeper analysis reveals that many of these techniques rely on tasks with specific preemptive characteristics. Although their statistical and computational results are satisfactory, their proposed solutions often overlook energy efficiency, carbon emissions, and VM and workflow application processing ratios' reliability. This represents a significant research gap in enhancing processing capabilities during the processing period.

While some techniques adopt metaheuristics for task processing ratios in cloud systems, they often fail to address crucial outcomes such as energy efficiency, carbon emissions, and scheduling reliability for both VMs and multiple applications in a workflow. Some methods use single server queuing systems for application scheduling, but this leads to extra computations during workflow application offloading, which burdens task migration and computation. Strategies like MSISCSOA [14], SDAR-NFAT [15], DCOHHOTS [16], A2C [17], DQN [18], and EBABC-PF [19] follow a similar concept but encounter issues related to extra computations and communication complexity. To address these limitations and provide a reliable workflow scheduling solution that is efficient in terms of energy and carbon emissions, we propose the A3C-CGT model.

Table 1 compares the proposed algorithm and existing methodologies qualitatively.

Table 1
A summary of the techniques commonly used in conducting a literature review.

Ref.	Paradigms	Strengths	Weaknesses
[20]	Hybrid particle swarm optimization (HPSO) algorithm	Minimize two objectives simultaneously: the total tardiness and the cost of electric power	The optimized preferred utilization range for all computing servers is overlooked. It is unable to identify the best utilization levels and anticipate the system's peak efficiency-to-utilization ratio.
[21]	Improved beetle swarm optimization (IBSO) algorithm	The optimization process aims to minimize power consumption, reduce resource wastage, and ensure compliance with service level agreements	Network link failure rate and dynamic characteristics of resource utilization are overlooked, mitigating the risk of a network bottleneck.
[22]	Evolutionary game theory virtual machine algorithm	The energy efficiency of a data center is maximized by adjusting the frequency of the processor	Crucial parameters are overlooked, such as dynamic task changes, different types of virtual machine instances, and overhead (e.g., idle overhead) resulting from adjacent VMs in network placement systems.
[23]	Energy-traffic-aware ant colony optimization (ETA-ACO) algorithm	The energy efficiency of a data center is optimized by adjusting the consumption of network bandwidth resources	Scheduling reliability is not acknowledged. Additionally, memory-intensive, input/output-intensive tasks, and migration costs are overlooked.
[24]	A power-aware cloudlet scheduling (PACS) algorithm	Decrease power usage, minimize operational expenses, and meet SLAs	Fail to consider preemptive characteristics for VM migration, which leads to hosting numerous applications on a single computing node.
[25]	Massive data similarity statistics analysis algorithm (MSSA)	Optimize resource utilization, minimize scheduling overhead, and reduce the makespan	The absence of fault tolerance mechanisms limits scalability and diminishes task scheduling efficiency.
[26]	Multi-objective hybrid genetic algorithm (MOHGA)	Satisfy end-to-end performance requirements while achieving energy efficiency and cost-effectiveness	Window matrix settings are not defined for allocated VM time slots. This causes VM overlapping and an increase in idle overhead between adjacent VMs.
[27]	Chaotic non dominated-sorting owl search algorithm (COSA)	Minimize the makespan, cost, and energy consumption while satisfying the given deadline and budget constraints	Integral factors, such as workload data size, host type, and VM overhead, are ignored and can all significantly impact the performance of the system.
[28]	Deadline constrained energy-aware workflow scheduling (DEWS) algorithm	The energy efficiency of cloud resource maximize by adjusting the processor's frequency	The preferred CPU utilization range is neglected, while lowering the CPU frequency can negatively impact the system's performance.
[29]	Critical parent reliability-based scheduling (CPRS) algorithm	Enhance the dependability of servers and network connections while overseeing application timelines, service quality, and financial limitations	Failure to investigate the energy function thoroughly significantly influences the system's effectiveness or process under examination.
[30]	Multi-objective hybrid bat (MOHBA) algorithm	Improve energy efficiency, reduce costs, shorten project duration, and optimize resource usage	Dynamic upper and lower thresholds are not considered per VM, causing overloaded and underloaded computing servers to be overlooked.
[19]	Enhanced binary artificial bee colony based pareto front (EBABC-PF)	Exhibits greater efficiency than the interquartile range and local regression, showing the ability to process vast quantities of data rapidly	Adaptable thresholds that can adjust based on the particular characteristics of the computing environment are overlooked.
[18]	Deep Q-Network reinforcement learning (DQN)	Simultaneously optimize two objectives: total operational costs and end-to-end delay makespan	The type of application running in the VMs is neglected, leading to potential increases in operational costs.
[17]	Advantage actor critic (A2C) reinforcement learning	Effectively reduce energy and carbon emission costs during workload scheduling	It cannot identify the optimal utilization levels or anticipate the system's peak efficiency-to-utilization ratio.

3. System design model

Fig. 1 [31] illustrates our integrated A3C-CGT paradigm for reliability-aware workflow scheduling in cloud systems comprising the application, scheduler, and data center layers.

In the application layer, cloud users can submit their applications (i.e., workflows) from various locations. The scheduler layer connects cloud users to the underlying cloud network hardware. Here, the A3C-CGT paradigm efficiently manages multiple VMs on fewer computing hosts and supports the dynamic activation and deactivation of VMs. The servers utilize the hardware infrastructure to create VM resources that meet service requests. A3C-CGT optimizes computing resources to maintain service quality while significantly reducing energy and CO2 consumption, contributing to environmental sustainability. The data center layer categorizes processing servers into three main categories based on two thresholds: overloaded server (OVS), regular state server (RSS), and underloaded server (UNS).

Each server's capacity in the data center is initially evaluated against these thresholds: $\tau_\alpha, \tau_\beta, \tau_\delta$. If a server's CPU utilization, U_{cpu}, or memory utilization, U_{mem}, exceeds τ_α, it is classified as OVS. If a server's U_{cpu} or U_{mem} is below τ_β, it is considered RSS, and its application allocation remains unchanged. However, if a server's U_{cpu} or U_{mem} falls below τ_δ, it is classified as UNS.

A3C-CGT consolidates VMs from OVS and reallocates them to UNS, reducing the number of active computing hosts while surplus inactive hosts enter a sleep state. The A3C-CGT model operates through five distinct stages.

3.1. Cloud architecture patterns

This section explores two key cloud architectures: workflow applications and network hardware models. The workflow application model addresses user needs and requirements, while the network hardware model focuses on the essential hardware characteristics of cloud servers and virtual machines.

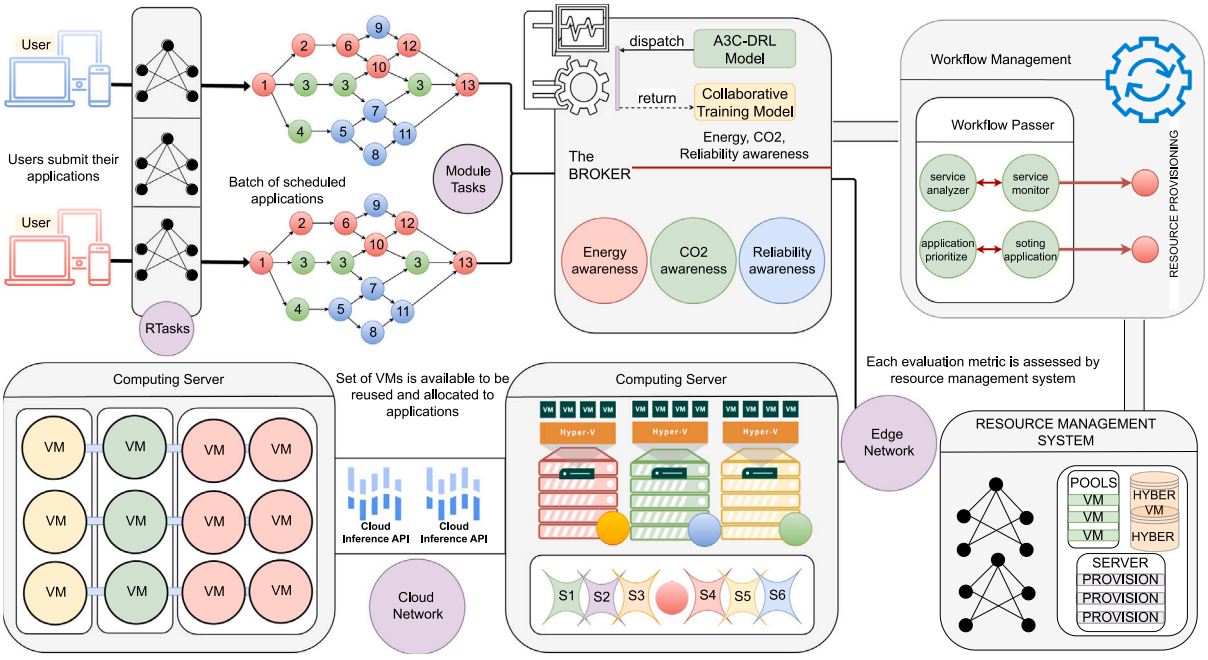


Fig. 1. The A3C-CGT system design model.

3.1.1. Workflow application model

The upper side of Fig. 2 [32] illustrates the directed acyclic graph (DAG) structure of the workflow formulated by our algorithm. This structure comprises a pair of elements: the count of user requests and the number of connections between different applications. We represent this concept mathematically in the following way:

$$\mathrm{DAG} = \left\{ \mathrm{R,E} \right\} \tag{1}$$

Here, $\{\tau_1, \tau_2, \ldots, \tau_n\}$ denotes a series of queued requests submitted by users, pending allocation to VMs. The dependency set of edges, represented as $\{e_1, e_2, \ldots, e_m\}$, describes the relationships among these requests. In our study, the quantity of data transmitted by an application to its neighboring applications is denoted by $(\omega_{s,d})$.

$$\mathrm{R} = \left\{ \tau_1, \tau_2, \ldots, \tau_n \right\}, \quad \mathrm{E} = \left\{ e_1, e_2, \ldots, e_m \right\} \tag{2}$$

The term $\omega_{s,d}$ on $e_{s,d}$ symbolizes the data transformation process from one specific application to another within the DAG, where $\omega_{s,d}$ represents the amount of data transferred. An application begins processing only after receiving all necessary instructions and data weights from preceding tasks. This initiates the computing sequence, whose complexity is a function of the task's cumulative and complexity-normalized computational demands.

However, in practical scenarios, an application's complexity is somewhat abstract, largely dependent on the computational intensity of the task's function and the specifics of the algorithm's implementation. The original complexity calculation is substituted with an input data aggregation function defined in the computing task to streamline this complexity function. For instance, when an application is mapped to a particular server and processes successfully, it forwards data and weight to successor applications. Additionally, in cases where an application has multiple starting or ending points, a virtual application with zero complexity is linked to all initial or final tasks involving transfers of zero-sized data.

3.1.2. Network hardware model

The lower side of Fig. 2 [32] depicts a hypothetical cloud network graph characterized by two computational elements: servers and communication links. These cloud servers are equipped with significant computational capabilities.

$$\mathrm{CNG} = \left\{ \mathrm{S,L} \right\} \tag{3}$$

The symbol S represents the collection of processing servers, while L denotes the communication link network interconnecting these servers. A specific failure rate defines every server and its corresponding link. The attributes of each server include its computing power and failure rate, whereas each link is characterized by its bandwidth, delay cost, and failure rate. Additionally, each server can host a finite number of VMs in this configuration.

3.2. Reinforcement learning approach

Reinforcement learning (RL) is ideal for settings where the environment is dynamic and workloads, resources, and priorities often change. Over time, RL learns to devise optimal or near-optimal scheduling strategies that enhance resource utilization and task efficiency. In addition, it effectively manages complex task dependencies and constraints, making it suitable for workflows with multiple interdependent tasks. RL can be applied in various workflow scheduling scenarios, from the smallest of operations to large distributed systems, because it provides the flexibility to deal with different grades of complexity. It also deals with uncertainty in workflows because of unplanned task durations or resource unavailability through learning and updating new circumstances.

3.2.1. RL limitations in solving workflow scheduling

RL has been increasingly applied to the scheduling of complex challenges, as it can learn from environmental interactions and adapt to changing conditions. However, there are several problems with applying RL to workflow scheduling:

- Trade-off exploration and exploitation: The RL model should balance exploring new workflow mapping strategies with exploiting proven,

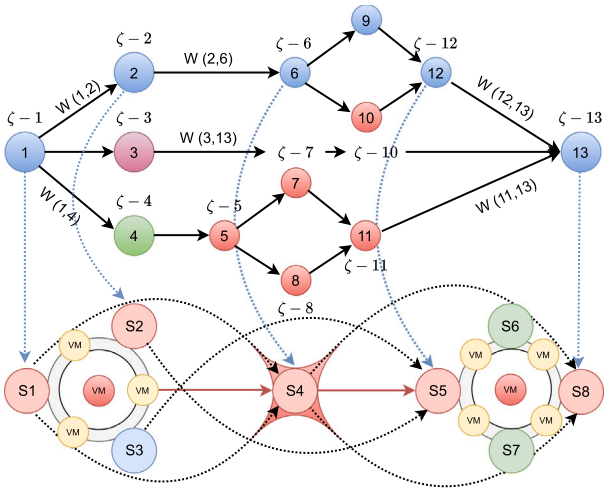


Fig. 2. The workflow and network graph models.

successful ones. This balancing act is not trivial, and poor management can lead to less effective scheduling decisions. It would be important to maintain this balance for effective workflow scheduling, especially in complex and changing environments. If enough exploration is done, the RL model may settle for suboptimal scheduling policies. On the other hand, excessive exploration may lead to inefficiency and long convergence times.

- Workflow convergence issues: In the RL, an optimal or near-optimal solution can be challenging, especially when dealing with intricate and nonlinear environments such as workflow scheduling. Thus, the RL model can reach a local optimum but not a global one, which entails making weak decisions in scheduling workflows.
- Workflow overfitting: An RL model trained explicitly on a specific type of workflow may overfit the conditions of that particular type, reducing its generalization to other workflows with different characteristics. The model would also not perform well in new types of workflows without much retraining or fine-tuning; it might be much less applicable in its effect.
- Workflow scalability challenges: Workflow scheduling, especially in the cloud or large-scale distributed systems, often involves a high-dimensional action space in many states. Traditional RL algorithms might have difficulty scaling to such problems because they require lots of computation and memory. The RL model might become slow to train or even unmanageable for a large-scope scheduling problem.
- Sparse and delayed rewards: Rewards are often sparse and delayed in such cases, as with workflow scheduling, where rewards take the form of reduced energy and carbon emissions or improved scheduling reliability. The RL agent could need to take action before being provided with critical feedback, which makes associating definite actions with consequences quite hard. Delayed and sparse rewards can make learning inefficient, with very slow progress on optimizing the scheduling policy.
- Complex environment dynamics: Workflows are subject to an immense degree of environmental dynamics that are very complex and often hard to predict, such as drastically changing workloads, resource availability, and execution dependencies. Here, the main challenge is accurately modeling and predicting such dynamics in light of a non-stationary, changing environment. The RL model can adapt slowly, resulting in poor scheduling in changing environments.
- Reward function formulation: The design of successful RL approaches lies in the reward function design, which guides the learning process. In other words, designing a reward function that correctly represents the goals of workflow scheduling, such as minimizing energy consumption and carbon emissions or increasing scheduling reliability, may be pretty tricky. A poorly designed reward function can push the agent to optimize the wrong things, which may cause harmful or counterproductive scheduling policies.
- Training time and computational resources: RL usually requires a lot of time for an effective training workflow alongside computational resources—especially in complex environments with rich state and action spaces. Therefore, the training process takes time and is not applicable in practice when workflows are scheduled in real-time or near real-time. On the other hand, high computational resource requirements can also limit RL's application in resource-constrained settings.
- Quality of simulation environment: An RL model is as good as the quality of the simulation environment used during training; a poorly simulated environment for training will result in a non-generalizing policy. Such discrepancies between the environments used during training and implementation will lead to undesirable performance when the developed RL model is implemented.

3.2.2. *A3C model for addressing workflow scheduling*

Although RL has shown promise, certain limitations remain that must be overcome through careful evaluation and possibly integrating other optimization methods to address workflow scheduling problems. Improving the effectiveness and practicality of RL in the context of workflow scheduling must focus on effective reward function design, scalability, and interpretability. To this end, we adopt the asynchronous advantage actor–critic (A3C) reinforcement learning approach for scheduling workflows across heterogeneous cloud networks.

Conventional A3C: The traditional A3C method encounters problems associated with learning from dynamic and complicated policy environments. All these trigger massive interaction with the environment for effective learning, hence computation costs and wastage of time. In addition, it becomes unstable during the training phase due to non-deterministic updates and ineffective coordination among the various parallel agents. The algorithm also has performance problems due to the communication overhead between the agents and the central learner. Furthermore, it is susceptible to hyperparameter settings; hence, it requires very cautious tuning to achieve good performance, which makes the process even more cumbersome and time-consuming. Also, traditional A3C is computationally intensive due to its reliance on parallelism and multiple environments, intensifying demands on computation resources in complex settings. These shortcomings have led to the development of an improved version of the A3C model to improve stability, efficiency, and scalability.

Improved variation of A3C: These limitations were addressed by developing an improved version of the A3C model. This new method integrates the two essential parts of the A3C model: the actor-network sends the state of the workflow application to an action space in which scheduling and processing occur, and a critic network judges the actions of the actor-network. This asynchronous operation technique contributes to the actor network being evaluated in parallel over several threads, thus making such threads capable of estimating losses inside the actor network while interacting with the global network by accumulating gradients post-runtime.

The improved A3C process brings in the adoption of residual convolutional neural networks that can capture complex dependency structures between the batch of workflow applications and allocated VMs. This speeds up the training process and makes the scheduling decision-making more effective. Two-dimensional folded data is passed into the actor–critic network and then projected into a one-dimensional format before being projected further into a fully connected hidden layer. This hidden layer has 256 neurons and is sized at three kernels with a step size of 0.5. Subsequently, the data passes through the hidden layer to connect with a softmax activation function that normalizes output to a range between 0 and 1. The network can multi-thread so that a different thread can supervise each agent independently. Thus, its training speed becomes much faster if multiple agents run simultaneously [33].

Setting A3C environment: Building on the work of [33], this study defines the scheduling time as S_τ, the state space as S_p, the action space as α_s, the state space sequence as S_{p+1}, and the generated reward as $\mathcal{R}_\omega$. Once the reward is generated, the procedure's policy, represented as η, assesses whether the reward conforms to the scheduling policy. This relationship is represented as $\langle S_p, \alpha_s, \mathcal{R}_w, \eta \rangle$.

- State space: The state space in the defined tuple consists of a batch of states $\mathcal{S} = \{S_1, S_2, \ldots, S_p\}$, representing a set of applications. Assume that $\mathcal{S}^{TaT}$ contains two data types. The first type reflects the dynamic nature of a computing node, defined as D_n, while the second type represents the dynamic nature of an application, defined as D_α [33].
- Action space: The action space in the defined tuple assigns a batch of workflow applications to compute VMs using the policy η. Mathematically, it is expressed as $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_s\}$. In this context, $\alpha_s = d_{i,j}$, where $d_{i,j}$ represents decision variables at the interval $\Delta\tau$ [33].
- Policy: The predefined policy η controls the payoffs resulting from the reward function, represented as $\eta(\alpha_s/S_p)$. This policy uses a neural network, expressed as $\eta(\alpha_s/S_p; \zeta_\alpha)$ [33].
- Reward function: One of the key parameters in the reinforcement learning technique is the reward function. This function model allows the system to calculate the reward obtained from the action space. It is expressed as $\mathcal{R}_w(S_p, \alpha_s)$. Mathematically, it can be formulated as follows [33].

$$\mathcal{R}_w = min(E_\tau, C\mathcal{O}_2), max(R(\underset{\forall i \in I}{\mathcal{W}}, \underset{\forall j \in J}{S})) \tag{4}$$

The reward function should provide an outcome; if the reward is negative, it needs to yield a cumulative discounted reward. Mathematically, it can be expressed as follows.

$$\mathcal{G}_w = \mathcal{R}_{w+1} + \psi.\mathcal{R}_{w+2}, \ldots, \psi^{\tau-\mathcal{T}-1}.\mathcal{R}_w \tag{5}$$

- Value function: The value function represents the expected outcomes of state and action sequences determined by the action and state spaces. The state value and action are computed using Eqs. (6) and (7), respectively. The expectation derived from these two functions is represented by E_p^λ [33].

$$\mathcal{V}^\lambda(S_p) = E_p^\lambda[\mathcal{R}_w + \psi.\mathcal{R}_{w+1} + \psi^{\tau-\mathcal{T}-1}.\mathcal{R}_w|S_p] \tag{6}$$

$$\begin{aligned}\mathcal{Y}^\lambda(S_p, \alpha_\tau) &= E_p^\lambda[\mathcal{G}_w|S_p, \alpha_\tau] \\ &= E_p^\lambda[\mathcal{R}_w + \psi^\lambda(S_{p+1}, \alpha_{\tau+1}), \ldots, |S_p, \alpha_\tau]\end{aligned} \tag{7}$$

Residual convolutional neural networks can also address the value function using the following network parameter, ϖ_α [33].

$$\mathcal{V}^\lambda(S_p) \approx \mathcal{V}(\alpha_\tau; \varpi_\alpha), \mathcal{Y}^\lambda(S_p, \alpha_\tau) \approx \mathcal{Y}(S_p, \alpha_\tau; \varpi_\alpha) \tag{8}$$

- Network training: In the improved A3C approach, multiple threads run simultaneously, each consisting of various agents that update decisions in the global network. This process continues until all iterations are completed, yielding maximum reward values. Initially, agents operate with their sample patterns in a network with multiple threads, aiming to observe reward gains so that the gradient can be assembled and elevated to the global network. The global network then compares the predicted and actual values to guide each agent in different threads, enabling efficient decision-making for workflow scheduling [33].

During the training phase, the policy function $\lambda(\alpha_\tau|S_p; \zeta_\alpha)$ is combined with the value function $\mathcal{Y}(S_p, \alpha_\tau; \varpi_\alpha^\sim)$, where ζ_α and $\varpi_\alpha^\sim$ are network control parameters. This combination checks the predicted reward. The gradient outcomes may vary when executing the same policy function per iteration. Therefore, the gradient ascent approach is adopted to achieve a cumulative gradient, mathematically formulated as follows [33]:

$$\bigtriangledown\zeta_\alpha E_p^\lambda[\mathcal{Y}^\lambda] = \bigtriangledown\zeta_\alpha \log\lambda(\alpha_\tau|S_p; \zeta_\alpha)\mathcal{Y}^\lambda \tag{9}$$

Gradient ascent is utilized to optimize the reward, thereby increasing the gradient value. The probability of the gradient being greater than or equal to zero is crucial; otherwise, it may decrease the learning rate. A3C employs the advantage function, $adv(S_p, \alpha_\tau)$, which guides the actor network's inefficient workflow scheduling without hindering the learning process. This function helps the system optimize converging biases during the solution selection process. Mathematically, it can be represented as follows [33]:

$$\begin{aligned}d\zeta_\alpha &= d\zeta_\alpha + \bigtriangledown\zeta_\alpha \log\lambda \log\lambda(\alpha_\tau|S_p; \zeta_\alpha) adv(\alpha_\tau, S_p) \\ &= d\zeta_\alpha + \bigtriangledown\zeta_\alpha \log\lambda \log\lambda(\alpha_\tau|S_p; \zeta_\alpha)(\mathcal{Y}(S_p, \alpha_\tau; \varpi_\alpha^\sim)) \\ &\quad - \mathcal{V}(S_p; \varpi_\alpha^\sim)\end{aligned} \tag{10}$$

Eq. (11) handles the update calculation for each agent from S_p to S_{p+1}. For instance, this includes the reward function $\mathcal{R}_w$ and the value function $\mathcal{V}(S_p; \varpi_\alpha^\sim)$. The temporal error is calculated according to Eq. (12) following the updated value function [33].

$$\mathcal{V}(S_p; \varpi_\alpha^\sim) = \mathcal{V}(S_p; \varpi_\alpha^\sim) + \mu(\mathcal{R}_w + \psi.\mathcal{V}(S_{p+1}; \varpi_\alpha^\sim) \tag{11}$$

$$d\varpi_\alpha^\sim = \frac{[\mu(\mathcal{R}_w + \psi.\mathcal{V}(S_{p+1}; \varpi_\alpha^\sim) - \mathcal{V}(S_p; \varpi_\alpha^\sim))]^2}{\mu\varpi_\alpha^\sim} \tag{12}$$

- Parameter updating: This process is a crucial step in the proposed model, as it continuously gathers data and allocates workflow applications to computing nodes using the improved A3C model, thereby increasing the reward values. Mathematically, it is expressed as follows [33]:

$$\zeta_\alpha = \zeta_\alpha + \mu.d\zeta_\alpha, \varpi_\alpha = \varpi_\alpha + \xi\varpi_\alpha \tag{13}$$

3.3. Coalitional game theory

A game-theoretic coalitional approach has been adopted to understand better the interaction of cloud entities—namely, providers with customers. Coalitional game theory studies how groups of players, also known as coalitions, work together and pool their resources to achieve shared goals.

Unlike classical game theory, where competition is prevalent among individual players, coalitional game theory carries out a detailed analysis of the advantages of forming coalitions and how the collective payoff is divided among the coalition members. A coalition's vitality lies in its players' cooperation and coordination. Three key concepts are essential when players are involved in these games.

The first key is the Core: a set of payoff distributions for which no subset of the players is incentivized to form a new coalition that would break away from the current coalition and under which no player could receive less than by doing the same. Another key but a second important one is the Shapley value in cooperative game solution concept, specifically in coalitional game theory, that provides one unique way of distributing among players the overall payoff while respecting the marginal contribution of each player toward the coalition, given all possible coalitions that can be formed. The third key is stability in a coalition—it must be autonomous, with no members having incentives to leave.

3.3.1. Merge-and-split based approach

Merge-and-Split Coalitional Game Theory (MαSCGT) essentially focuses on coalitional creation and reformation among the players or servers in meeting maximization objectives such as energy efficiency, CO2 reduction, and reliability. It includes the merger and splitting of coalitions so that dynamic changes can be done with the interactions between players and interrelated goals. The basic ideas in it are:

- Coalitions: A coalition is a set of processing nodes working together to minimize the consumed energy and emitted CO2 while maximizing their performance-to-power ratio (PPR).
- Merging: It is done by merging two or more coalitions into a larger one, which is helpful for all the providers. For instance, two smaller server groups can come together to save on energy consumption, reduce CO2, and be more reliable with better performance. Then, they will form a more effective, bigger coalition.
- Splitting: If one group of servers is not enough or powerful enough to perform those actions, they could divide into other coalitions, ensuring the adaptation in the context of changing conditions and objectives. The principal aspiration of this strategy is that it raises the payoff for every coalition.
- Payoff: The payoff is incrementing the scheduling reliability by reducing energy and CO2 consumption.

This two-pronged method provides greater flexibility and effectiveness with decision-making, allowing for the selection of machine nodes that can (a) lower the failure rate and (b) achieve the highest PPR. We model this selection problem as a coalitional game defined by $\eta = (S, \zeta)$.

- S designates a finite set of servers (i.e., players) in the cloud domain; and
- $\zeta : 2^S \rightarrow \omega$ (characteristic function) Each coalition $s \subseteq S$ is associated with an actual number representing the payoff $\omega(S)$ that the members of the coalition can distribute among themselves. However, our algorithm assumes that $\omega(\phi) = 0$, meaning that an empty alliance has no payoff.

3.3.2. MαSCGT functionality

During this phase, processing servers are trained to determine their ideal utilization levels under various workloads. "Preferred utilization" refers to the optimal operating point where servers achieve maximum efficiency, considering power consumption, CO2 emissions, and scheduling reliability.

To manage server performance effectively, the method aggregates each server's preferred utilization levels. These levels are then merged and split to create a consolidated dataset known as "optimal utilization". Combining the preferred utilization rates of all servers allows each server's most efficient utilization level to be selected. The focus is splitting the utilization levels to achieve higher service efficiency (SE). The goal is to establish an efficient collaboration that ensures servers operate optimally, enhancing overall utilization across the network. The range $[\omega, \omega + \eta]$, where η represents a predefined utilization range, typically defines the scope of this optimal utilization. This range is essential in setting the operational parameters for optimal server performance.

Calculating ω, the starting point of this optimal utilization range, involves a specific approach. This method analyzes historical data, current server performance metrics, and predictive models to identify each server's most effective operating point. By calculating and applying ω, the proposed approach ensures that servers function within a range that maximizes efficiency while meeting the constraints and requirements of the cloud computing environment [34].

$$\omega = argmax \int_{v}^{v+1} \left(F_U^u\right) du, \quad \text{s.t. } 0 \le \omega \le \omega + 1 \le 1 \tag{14}$$

The function model, expressed as F_U^u, symbolizes the chosen usage based on the desired utilization levels, with u being a value that varies between 0 and 1.

We aim to maximize the value of ω to increase each coalition's payoff. Therefore, Inspired by [35], we compute the value of ω as follows:

$$\omega = \frac{1}{\mathcal{Y}} \sum_{\alpha=1}^{\mathcal{Y}} \mathcal{B}U_\alpha \quad \text{s.t. } U_{\mathcal{L}} \le \mathcal{B}U \le U_{\mathcal{H}} \tag{15}$$

3.3.3. Impact of MαSCGT on A3C improvement

Incorporating merge-and-split coalitional game theory into A3C can enhance the learning efficiency and performance of the agents by enabling them to form and dissolve coalitions according to their common objectives and rewards. This game theory approach can improve the A3C algorithms through the following processes.

- Coalition formation: Environmental A3C agents assess potential coalitions by analyzing their current states, goals, and expected rewards. These coalitions are established to optimize collaborative actions that maximize cumulative rewards.
- Merge and split operations: When agents can attain greater rewards collectively, they merge their coalitions, which is especially beneficial in complex environments where collaboration is crucial. Conversely, they split when the coalition's effectiveness diminishes or when individual agents can earn more rewards by acting independently.
- Learning process: As agents interact, they continuously update their policies (actor) and value functions (critic) in response to the evolving coalitional dynamics. The A3C method gains an advantage from the fluid nature of coalition formations, as it helps avoid local optima by exploring various strategies through coalition mergers and splits.
- Reward distribution: When a coalition successfully reaches a goal, the rewards are distributed among the members according to their contributions using a fair division rule, such as the Shapley value. This approach motivates agents to join coalitions where their contributions are most valued.

3.3.4. Flowchart illustration

Fig. 3 shows the integration of the merge-and-split coalitional game theory with the A3C method. This chart shows the basic steps from initialization of agents into the environment, exploration of the environment, formation of coalitions, making decisions to merge or split, updating of the actor–critic models, and going through iterations until it converges. This integration makes the learning efficiency of the A3C method even better, allowing the agents to form and dissolve coalitions dynamically, thereby optimizing strategies for better overall performance. The major steps in doing so are given below:

1. Agent initialization and environmental exploration: Agents with a random goal-achieving policy start randomly searching the environment.
2. Identification of potential coalitions: The agent further explores the environment to identify another agent with similar goals to form potential coalitions.

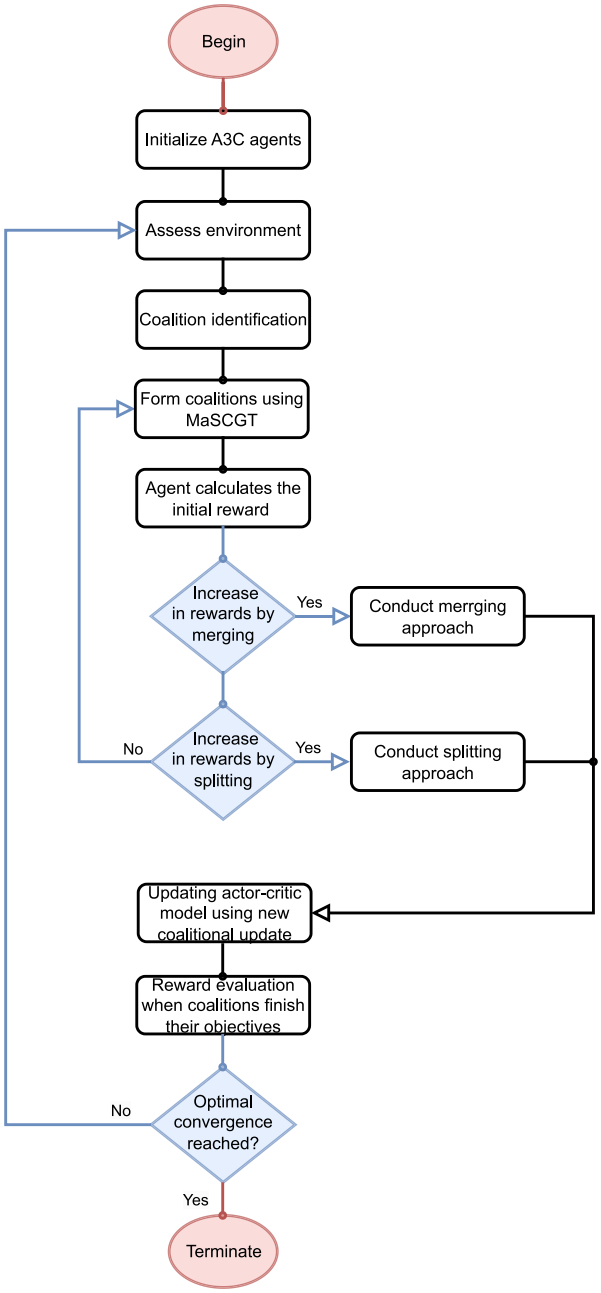


Fig. 3. A3C-CGT flowchart design.

3. Merge or splitting decision: It decides on a merge or splitting when two or more coalitions are formed, and other coalitions that aim at the same goal can optionally be selected.
4. actor–critic updating: Every agent updates its policy and value function with fresh coalitional data from the scheduler.
5. Reward distribution assessment: The coalitions reach a goal and share a total reward received according to their contributions.
6. Continue this process until convergence: The above process will be iterated until the agents can converge policies toward an optimal strategy.

3.3.5. Illustrative analysis

Fig. 4 illustrates the implementation of merge and split operations within coalitional game theory. In this coalitional game, players collaborate to reduce energy consumption and CO2 emissions while improving scheduling reliability. The merge and split operations are crucial for understanding how the value of coalitions changes when players unite or separate. The process involves three key steps. The first step is initialization, followed by the merging and splitting operations.

Assume a set of players $n = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$. Each player assigns an initial value to participate in the game. These players form two separate coalitions, S_1 and S_2. Coalition S_1 includes (p_1, p_2, p_3), while coalition S_2 consists of (p_4, p_5, p_6, p_7). We then analyze the impact of merging these two coalitions into a single coalition as follows:

- Coalition $S_1 = \{p_1, p_2, p_3\}$
- Coalition $S_2 = \{p_4, p_5, p_6, p_7\}$

We calculate the values of coalitions before the merging operation as follows:

- $\omega(S_1)$ is the value that coalition $\{p_1, p_2, p_3\}$ can achieve.
- $\omega(S_2)$ is the value that coalition $\{p_4, p_5, p_6, p_7\}$ can achieve.

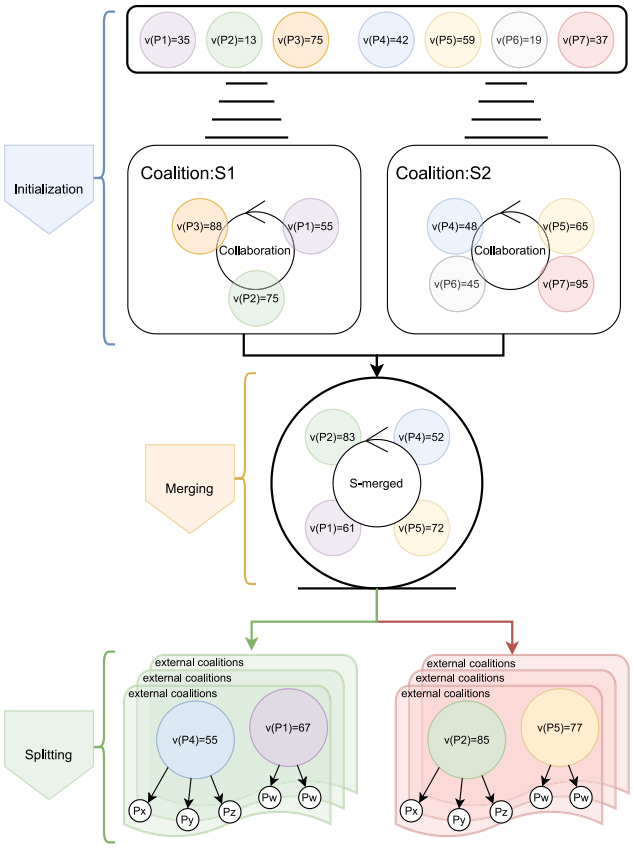


Fig. 4. The process of merging and splitting approach.

We then proceed to compute the coalition values after the merging process as follows: Initially, we consider a merged coalition: $S_{\text{merged}} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$.

Afterward, we calculate the value of S-merged, represented as v(S-merged). Lastly, we compare the values: the sum of the individual values, $\omega(S_1) + \omega(S_2)$, with the value of the merged coalition, $\omega(S_{\text{merged}})$. In our example, we have:

- $\omega(S_1) = \{p_1, p_2, p_3\} = (55, 75, 88) = 218$
- $\omega(S_2) = \{p_4, p_5, p_6, p_7\} = (48, 65, 45, 95) = 253$

After merging the most capable players into one coalition, the result is as follows:

$S_{\text{merged}} = \{p_1, p_2, p_4, p_5\} = (83, 52, 61, 72) = 268$.

Now, compare S_{merged} with $\omega(S_1)$ and $\omega(S_1)$ separately to assess the effectiveness of the merging operation. The merged coalition's value is 268, which is higher than the values of both S_1 and S_2. This indicates that merging can offer additional benefits, such as synergies or more efficient resource utilization. Subsequently, p_1 and p_4 from the S_{merged} coalition attempt to split and merge with other coalitions outside the S_{merged} domain to enhance their effectiveness further. They join external coalitions and collaborate with other players to improve and increase EUR values.

These explanations aim to clarify the effects of merging and splitting coalitions in game theory, which is vital for designing and evaluating cooperative strategies.

4. Mathematical model

This section explores the mathematical models applied to formulate essential evaluation metrics used in the A3C-CGT paradigm. Table 2 provides definitions for the symbols and abbreviations used throughout this paper.

4.1. Energy consumption model

This study examines two significant types of cloud applications: memory-intensive and CPU-intensive. The scheduling heuristic introduced here identifies the most energy-consuming resources and directs suitable placement or mapping strategies for VMs.

The total power consumption of a physical server has two components: fixed power consumption (P_c), which stays constant regardless of VM activity, and dynamic power consumption (P_d), which varies with the power used by running VMs [36,37].

$$P_t = P_c + P_d \tag{16}$$

Given the significant influence of the CPU on energy consumption, we devised a DVFS strategy to manage CPU frequency. This method enables precise adjustments to the CPU clock frequency and supply voltage, considerably improving our approach and leading to substantial energy savings. Thus, we have organized the power model as follows [38]:

$$P_t = P_c + \eta \left(P_{act} + \mathcal{V}^{\tau} \mathcal{Y}_{dd}^2 f^{\alpha} \right) \tag{17}$$

where P_{act} represents the active power expenditures, and $\mathcal{V}^{\tau} \mathcal{Y}_{dd}^2 f^{\alpha}$ denotes the dynamic power expenditures, which significantly contribute to overall energy consumption. In this context, $\mathcal{V}^{\tau}$ is the effective loading capacitance, $\mathcal{Y}_{dd}^2$ represents the CPU voltage of the server, and f^{α} indicates the clock frequency. The relationship between CPU clock frequency and supply voltage is crucial, as lowering the frequency

Table 2
Primary symbols explained in this study.

Parameters	The explanation of key symbols Explanations	Expressed in mathematical Formulas
R	The series of queued requests submitted by users	Eqs. (1), (21), (23), (24), (35)
E	The relationships among these requests	Eq. (1)
S	The collection of processing servers	Eqs. (3), (21), (23), (24), (27), (28)
L	The communication link network interconnecting these servers	Eq. (3)
$\mathcal{R}_w$	The reward function in the RL approach	Eq. (4)
E_τ	The energy consumption of server S_τ	Eqs. (4), (20), (34)
$\mathcal{CO}_2$	The carbon emissions	Eqs. (4), (25), (34)
$\mathcal{W}$	The scheduled workflow with a set of applications	Eqs. (4), (23), (24), (29), (30), (35)
$\mathcal{R}$	The scheduling reliability of assigning a set of applications	Eq. (4)
F_U^u	The operational utilization based on the desired utilization levels	Eq. (14)
$\mathcal{BU}$	The preferred utilization utilize by allocated servers	Eq. (15)
P_e	The fixed power consumption of server S_j	Eq. (16)
P_d	The dynamic power consumption of server S_j	Eqs. (16), (17)
P_{act}	The active power expenditures of server S_j	Eq. (17)
$\mathcal{V}^\tau \mathcal{Y}_{dd}^2 f^\alpha$	The server's dynamic power expenditures	Eq. (17)
P_u	The power consumption of server S_j at utilization u	Eqs. (18), (19)
P_{cpu}	The power consumption of server S_j's CPU	Eqs. (18), (27)
P_{mem}	The power consumption of server S_j's memory	Eq. (18)
$P_{cpu,max}$	The maximum power consumption of server S_j's CPU	Eq. (19)
$P_{mem,max}$	The maximum power consumption of server S_j's memory	Eq. (19)
η_0	The server's initial frequency of faults	Eq. (21)
$S_{\upsilon_j,\mathcal{R}\alpha}$	The server's current operational frequency	Eqs. (21), (22), (23)
$S(\upsilon_j, \mathcal{R}min)$	The minimum operational frequency for the server	Eq. (22)
C_1	The computational complexity associated with the data input size	Eq. (23)
C_2	The total computational complexity	Eq. (23)
$\mathcal{COP}$	The efficiency of the collaborative system	Eq. (25)
δ	The constant power consumption	Eq. (25)
$\alpha_i \left(f_j^i\right)^3$	The variable power cost with the CPU operating frequency f_i	Eq. (25)
α_i	denoting the proportionality constant	Eq. (25)
$E_j^i(f)$	The execution time of a specific task using a given frequency	Eq. (25)
ξ_i	The number of concurrent data transfers on the link $\mathcal{L}_{i,j}$	Eq. (26)
$\mathcal{BW}_{i,j}$	The allocated bandwidth between the source and destination servers	Eq. (26)
$\mathcal{DL}_{i,j}$	The transferring delay of instructions from the source to destination	Eq. (26)
OP_e	The operational costs associated with server S_j	Eq. (27)
E_c	The cost of electricity	Eq. (27)
$C_x(j)$	The computed complexity of the assigned workflow	Eq. (29)
$\mathcal{V}m$	The total number of virtual machine	Eqs. (30), (31), (36), (37)
$\mathcal{V}_{i,cpu}$	The total CPU allocated to virtual machine $\mathcal{V}_i$	Eq. (32)
$S_{j,res(cpu)}$	The total available CPU of server S_j	Eq. (32)
$\mathcal{V}_{i,mem}$	The total memory allocated to virtual machine $\mathcal{V}_i$	Eq. (33)
$S_{j,res(mem)}$	The total available memory of server S_j	Eq. (33)
$\mathcal{O}_\mathcal{V}$	The overcrowded virtual machine	Eq. (36)
$U_\mathcal{V}$	The undercrowded virtual machine	Eq. (36)
$i\mathcal{QR}$	The inter-quartiles range	Eq. (38)
$\mathcal{Q}_u$	The upper quartiles	Eq. (38)
$\mathcal{Q}_l$	The lower quartiles	Eq. (38)

results in a corresponding reduction in supply voltage. Our power model is primarily based on the frequency parameter, represented as f^α, with α calculated as $1+2/\xi$, where ξ is at least 3. This model mainly considers the CPU and memory characteristics [37].

$$P_u = P_{cpu} + P_{mem} \tag{18}$$

Several studies have highlighted that approximately 70% of a host machine's power is consumed when it is idle compared to when it is fully utilized. This observation suggests that turning off an inactive host can yield substantial power savings and improved energy efficiency. Therefore, in our proposed meta-heuristic approach, we integrate the idle fraction into the energy model as follows [39].

$$\begin{aligned} P_u = &\left[\delta \times P_{cpu,max} + (1-\delta) \times P_{mem,max} \times u\right] \\ &+ \left[\delta \times P_{mem,max} + (1-\delta) \times P_{mem,max} \times u\right] \end{aligned} \tag{19}$$

where $P_{cpu,max}$ denotes the maximum power a host can consume while executing tasks that demand significant processing power on its CPUs. This value accounts for the highest power usage of the host when operating at total capacity [40]. Similarly, $P_{mem,max}$ refers to the maximum power a host can consume while handling tasks that require substantial memory usage. This value considers the peak power consumption of the host when it is at total capacity for memory-intensive tasks [41,42].

The energy consumption of the processing server can vary with changes in CPU and memory usage, meaning it fluctuates over time (i.e., it is a function of time). According to Eq. (20) [39], the total energy expenditure of a processing server can be represented by integrating the power consumption function over time.

$$E_\tau = \int_\tau^\infty P\big(u(t)\big)\, dx \tag{20}$$

4.2. Reliability efficiency model

Server breakdowns and link failures are common issues in task scheduling, occurring in both permanent and transient forms. Transient failures often have a more significant impact than permanent ones. Therefore, this paper primarily addresses transient defects directly affecting system productivity. Research has shown that failures in cloud resources are unavoidable and can significantly increase the total delay in scheduling processes. These failures are assumed to follow a Poisson distribution. Consequently, we consider the fault rate occurrence, denoted by η, which is heavily influenced by the operational frequency of the server [43–46].

$$\eta\Big(S\big(\upsilon_j, \mu\mathcal{R}\big)\Big) = \eta_0 \times \zeta\Big(S\big(\upsilon_j, \mathcal{R}\alpha\big)\Big) \tag{21}$$

where η_0 represents the initial fault frequency when the server, denoted as $S(\upsilon_j, \mu\mathcal{R})$, operates at its maximum frequency. The term $S_{\upsilon_j,\mathcal{R}\alpha}$ indicates the server's current operational frequency, while $\zeta\left(S(\upsilon_j, \mathcal{R}\alpha)\right)$ is associated with a function that decreases over time.

Our approach uses the exponential model first introduced in [43] and subsequently applied in studies [44,45], commonly referred to as Eq. (21). This specific equation outlines the relationship between the critical cost of the circuit and the variable η, reflecting the traditional exponential correlation described in Eq. (22).

$$\begin{aligned}\eta\left(S(\upsilon_j, \mu\mathcal{R})\right) &= \eta_0 \times \zeta\left(S(\upsilon_j, \mathcal{R}\alpha)\right) \\ &= \eta_0.10^{\frac{d(1-S(\upsilon_j,\mathcal{R}\alpha))}{1-S(\upsilon_j,\mathcal{R}min)}}\end{aligned} \tag{22}$$

In our framework, we define a steady, positive fault rate denoted by ζ, and $S(\upsilon_j, \mathcal{R}min)$ represents the minimum operational frequency for the server $S(\upsilon_j)$. We evaluate the reliability of workflow module applications in cloud environments based on [43–45]. In task scheduling, resource failures are typically statistically independent due to the involvement of various types of resources.

$$R\left(\underset{\forall i\in I}{\mathcal{W}}, \underset{\forall j\in J}{S}\right) = e^{-\eta\left(S(\upsilon_j,\mathcal{R}\alpha)\right)\times\frac{w\left(c_1,c_2\right)}{s\left(\upsilon_j,\mathcal{R}\alpha\right)}} \tag{23}$$

where $\mathcal{W}$ represents a subset of the applications within a workflow, C_1 refers to the computing complexity associated with the instruction weights, and C_2 represents the total execution complexity. The overall reliability of each module task is determined by calculating the product of the reliability of each application, as detailed below.

$$R(\mathcal{W}) = \prod_{i=1}^{n} R\left(\underset{\forall i\in I}{\mathcal{W}}, \underset{\forall j\in J}{S}\right) \tag{24}$$

This equation is crucial in our application dispatching process, as it ensures the reliability of each model when deployed on operational machines.

4.3. CO2 emissions model

Our study focuses on carbon dioxide emissions. This framework mainly depends on the efficiency of the collaborative system, the constant power consumption, the variable power cost associated with CPU operating frequency, and the execution time of a specific task at a given frequency [46].

$$\begin{aligned}C\mathcal{O}_2 &= \left(1+\frac{1}{C\mathcal{OP}}\right) \\ &\times \sum_{i}^{\mathcal{Y}}\left(\delta_i + \alpha_i\left(f_j^i\right)^3 \times E_j^i\left(f_i^{max}\right) \times \frac{f_i^{max}}{f_j^i}\right)\end{aligned} \tag{25}$$

where $C\mathcal{OP}$ symbolizes the efficiency of the collaborative system. δ_i represents the constant power consumption, while $\alpha_i\left(f_j^i\right)^3$ signifies the variable power cost associated with the CPU operating frequency f_i, with α_i denoting the proportionality constant. The variable $E_j^i(f_i^{max})$ represents the execution time of a specific task using maximum frequency.

4.4. Bandwidth dependency model

ξ_i represents the number of concurrent data transfers on the link $\mathcal{L}_{i,j}$ during the interval $\Delta\tau$. Additionally, let $\zeta_i(.)$ denote the amount of data transfer partially completed within the time frame $[\tau, \tau+\Delta\tau]$, with the condition that ξ_i remains unchanged. Mathematically, this concept is formulated as follows:

$$\mathcal{BD}\left(\xi_i, \mathcal{L}_{i,j}\right) = \frac{\sum_{i=1}^{m}\xi_i \times \zeta_i(.)}{\mathcal{BW}_{i,j}} + D\mathcal{L}_{i,j} \tag{26}$$

where $\mathcal{BW}_{i,j}$ represents the allocated bandwidth between the source and destination servers, and $D\mathcal{L}_{i,j}$ denotes the delay of transferring a batch of instructions from the source to the destination servers.

4.5. Service productivity model

Cloud owners have shown interest in enhancing this metric as it indicates a better balance between the service charge and the service operating expenses. The metric is defined in the following way:

$$SP = OP_c \times \sum_{i=1}^{n}\sum_{j=1}^{m}\left(\frac{\sum_{i=1}^{m}\xi_i \times \zeta(.)}{P\left(S_{j,\text{dvfs}}\right)}\right) - E_c \tag{27}$$

where OP_c refers to the operational costs associated with allocated servers, E_c signifies the cost of electricity, and $P(S_{j,\text{dvfs}})$ denotes the power of the server S_j adjusted using the DVFS technique.

4.6. EUR model

This metric evaluates the effectiveness of services rendered for completed jobs. It is calculated by dividing the total cost, which encompasses the extra time spent during the initiation, suspension, and termination phases, by the actual processing charge.

$$EUR = \frac{S_{j,\text{dvfs}} \times \left[dis(\mathcal{V})\right]}{S_{j,\text{dvfs}} \times \left[dis(\mathcal{V}) + \sum_{i=1}^{m}\xi_i \times \zeta(.)\right]} \tag{28}$$

where $dis(\mathcal{V})$ includes the initiation time of VMs, the suspension duration between the allocation of two consecutive VMs, and the reduced time experienced by reused VMs.

4.7. QoS efficiency model

This metric is crucial in task scheduling as it measures the cloud services provided during task processing. For example, when an application is executed at total capacity, the QoS metric increases significantly. A primary concern regarding QoS is the failure to meet a task's deadline. It is formulated as follows:

$$QoS = \frac{\sum_{j=1}^{n}\mathcal{W}_j \times C_x(j)}{\sum_{i=1}^{m}\xi_i \times \zeta(.)} \tag{29}$$

where $\mathcal{W}$ denotes an assigned workflow and $C_x(j)$ represents the computed complexity of the assigned workflow.

5. Problem statement

This section highlights three critical stages. First, the proposed problem is detailed. Second, the problem is formulated. Third, the conflicting objective is clarified, focusing on the objective function.

5.1. Problem description

The scheduling of workflows is part of a multi-objective A3C and CGT optimization problem, where ($\mathcal{V}m$) stands for virtual machines and ($\mathcal{W}$) stands for different kinds of workflows. A matrix of size ($\mathcal{V}m\times\mathcal{W}$) = ($\mathcal{V}+\mathcal{W}$) is used to divide workflow applications among the VMs and show where each task should be done, as shown in Eq. (30).

$$\begin{bmatrix} x_{1,1}x_{1,2} & \cdots & x_{1,n-1}x_{1,\mathcal{W}} \\ x_{2,1}x_{2,2} & \cdots & x_{2,n-1}x_{2,\mathcal{W}} \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ x_{\mathcal{V}m,1}x_{\mathcal{V}m,2} & \cdots & x_{\mathcal{V}m,\mathcal{W}} \end{bmatrix} \tag{30}$$

$$x_{i,j} = \begin{cases} 1, & \text{if } \xi_i \to \mathcal{V}m. \\ 0, & \text{otherwise.} \end{cases} \tag{31}$$

Each iteration within the scheduling cycle represents a potential solution that could lead to a feasible resolution. All objective functions undergo equal iterations to expedite the response process. There are two main problems that need to be fixed. The first is figuring out how to make a collaborative-based A3C-CGT reliability-aware workflow scheduling framework that can organize and balance different kinds of tasks across different VMs in the cloud. The second is developing ways to make this collaborative system less time-consuming so that it can be used in the real world.

5.2. Problem formulation

This section elucidates the integration of the merge-and-split-based approach with an advanced A3C to reduce complexity in workflow scheduling. Such an approach is essential in advancing the performance level of our paradigm, as it can properly regulate and bring stability into the operations of several VMs when handling various VM instance types. On a practical note, this approach can bear utility by gaining the best possible resource use for improved system performance. The bin-packing problem of locating appropriate destinations for migrated VMs does, however, pose a challenge. VMs are individual instances that must be packed onto as few active processing servers as possible.

Relocated VMs should be given enough CPU and memory resources to make composition with other VMs. That being the case, a processing server can only accept a migrated VM if the available resources outperform the requirements of the migrated VM. These issues are addressed in the proposed solution because it uses MαSCGT, A3C, and dynamic adaptive threshold SLA-QoS-RALOC algorithms to ensure better SLAs under QoS when cloud users' requirements shift from one scenario to another. Mathematically, it can be stated as follows:

$$\sum_{i=1}^{\mathcal{K}} \mathcal{V}m_{i,cpu} < S_{j,res(cpu)} \tag{32}$$

$$\sum_{i=1}^{\mathcal{K}} \mathcal{V}m_{i,mem} < S_{j,res(mem)} \tag{33}$$

The A3C-CGT represents the mapping of migrated VMs, and each computing node is modeled as a vector of the number of migrated VMs. Every segment of the vector is labeled with its weight, which is, in essence, the index of the executable apparatus chosen to host the interconnected VM. Hence, our workflow scheduling is achieved in Eqs. (34) and (35).

$$\min \sum_{j=1}^{m} \left(E_{\tau}, C\mathcal{O}_2 \right) \tag{34}$$

$$\max \sum_{i=1}^{n} \left(R(\mathcal{W}_i), \mathcal{BD}_{s,d}(\mathcal{L}_{s,d}), SP, EUR, QoS \right) \tag{35}$$

These are conflicting objectives, and it is essential to balance them by identifying all overloaded and underloaded resources in the following manner.

$$\mathcal{Y} = \sum_{i=1}^{\tau} \mathcal{V}_i, \text{ where } \mathcal{V}_i \in \left\{ \mathcal{O}_{\mathcal{V}} \vee U_{\mathcal{V}} \right\} \tag{36}$$

where $\mathcal{O}_{\mathcal{V}}$ represents the overloaded resources, and $U_{\mathcal{V}}$ denotes the underloaded resources. Consequently, Eq. (37) provides the mathematical representation of the objective function.

$$F(\mathcal{Y}) = \min \left\{ \mathcal{Y}_i \,\middle|\, 1 \le i \le \mathcal{V} \right\} \tag{37}$$

5.3. Contradicting multi-objective optimization

This paper has two objectives: energy consumption and reliability. In developing the A3C-CGT concept, other evaluation metrics such as CO2 emissions, QoS efficiency, service productivity, EUR, and bandwidth dependency have been considered to create a more effective model. The problem formulated in this paper is treating the subject of minimizing energy consumption but maximizing reliability efficiency as a multi-objective optimization problem (MOOP). The reason is that it is not possible to achieve both objectives simultaneously because they oppose each other.

We chose to work on these conflicting objectives because their effectiveness directly impacts the other evaluation measures: CO2 emissions, QoS performance, service productivity, EUR, and bandwidth dependency. The design and optimization issues related to balancing these conflicting objectives are delicate processes in which trade-offs must be made according to the system's specifications. For instance, one can allow greater energy consumption for mission-critical applications with higher reliability but choose to enhance energy efficiency for less critical applications.

- Energy consumption: Decrease the power used by computing servers, for example, by using energy-efficient algorithms or even turning parts of systems off when they are idle. Such measures go a long way in reducing the overall energy footprint, but they also tend to affect system performance and reliability.
- Reliability: High reliability usually requires redundancy, fault tolerance, and continuous operation, which can lead to greater energy consumption. For example, maintaining server and communication link failure rates, running multiple instances of VMs, or keeping systems fully operational to avoid downtime consumes more power.

6. Algorithm design

We divided the A3C-CGT framework for energy-efficient and reliable workflow scheduling for offloading applications to cloud resources into three separate algorithms:

6.1. MαSCGT algorithm

The main goal of Algorithm 1 is to establish a resilient cluster of computing nodes aimed at improving the PPR through the merge-and-split coalitional game theory (MαSCGT) algorithm. The algorithm categorizes servers into specific clusters based on their usage patterns.

The algorithm has two primary objectives: first, to form robust server clusters optimized for performance by executing predefined tasks while reducing energy consumption, and second, to determine the most effective way to operate these clusters. Furthermore, the process includes identifying the controller server with the highest PPR.

6.1.1. Functionality of MαSCGT steps

The first step of Algorithm 1 involves setting up a varied computing environment with multiple processing servers, each hosting several VMs. In Steps (1–3), two new metrics are introduced to assess load usage efficiency and identify hosts with the highest utilization rates.

Next, a loop is created to train computing servers under different utilization weights. This process involves classifying servers based on performance and selecting the leader with the greatest power in Steps (4–27). In Step (5), each server is assigned 11 different utilization levels, ranging from 0% to 100%. These levels are used to execute a set of tasks in Steps (7–9) to identify the utilization that ensures optimal performance with minimal energy consumption, known as "preferred utilization". Servers may have one or more preferred utilization levels. In Step (10), servers are grouped into sets, denoted as ($\mathcal{G}^*$), based on their efficiency in executing tasks quickly and with low energy

Algorithm 1: MαSCGT Algorithm

Input: *Cluster of computing servers hosting a restricted quantity of VMs*
Output: *Categorize processing hosts into clusters and choose a master cluster leader*

$en^* \leftarrow$ Create heterogeneity and connectivity completeness environment of group of processing servers.
Initialize $\mathcal{SW}^g$.
$\delta \leftarrow$ MAXVALUE.
for $\left(\forall \mathcal{S} \in en^*\right)$ **do**
  $\mathcal{S} \leftarrow$ Eleven effective utilization loads.
  Train each processing server at different loads to find the best pair server-utilization combination through:
  for $\left(\forall u \in \mathcal{R}^*[0-10]\right)$ **do**
    $\mathcal{SW}^g \leftarrow \mathcal{SW}^g + \left(\frac{\sum_{i=1}^{n} \delta_i(\tau) \times \zeta_i(\tau)}{\mathcal{P}_{\mathcal{S}}(u)}\right)$
  end
  $\mathcal{Q}^*_{\mathcal{S}} \leftarrow$ sort computing machines into various categories, $\mathcal{Q}^*_{\mathcal{S}^+}$, based on their capacities.
  $\mathcal{Q}^*_p \leftarrow$ sort utilization loads based on which utilization can get lower $\mathcal{SW}^g$ values .
  $\mathcal{Q}^*_{\mathcal{B}} \leftarrow$ a batch of utilization with the highest $\mathcal{Q}^*_p$ values.
  for $\left(\forall \mathcal{S} \in \mathcal{Q}^*_{\mathcal{S}}\right)$ **do**
    for $\left(\forall \mathcal{Q}^*_p \in \mathcal{S}\right)$ **do**
      if $\left(\left(\mathcal{Q}^*_p > \mathcal{Q}^*_{p,H}\right) \vee \left(\mathcal{Q}^*_p + \mathcal{V}(u) > 1\right)\right)$ **then**
        continue
      end
      else if $\left(\left(\mathcal{Q}^*_p + \mathcal{V}(u) > \mathcal{Q}^*_{\mathcal{B}} - 0.03\right) \wedge \left(\mathcal{Q}^*_p + \mathcal{V}(u) < \mathcal{Q}^*_{\mathcal{B}} + 0.03\right)\right)$ **then**
        return $\mathcal{S}$
      end
      else if $\left(abs\left(\mathcal{Q}^*_p + \mathcal{V}(u) - \mathcal{Q}^*_{\mathcal{B}}\right) < \delta\right)$ **then**
        $\delta \leftarrow abs\left(\mathcal{Q}^*_p + \mathcal{V}(u) - \mathcal{Q}^*_{\mathcal{B}}\right)$
        $\mathcal{S} \leftarrow \delta$
      end
    end
  end
end

consumption. Steps (11–12) create two queues, ($\mathcal{P}u$) and ($\mathcal{B}u$). ($\mathcal{P}u$) lists all servers' preferred utilizations as being most to least efficient, while ($\mathcal{B}u$) includes only the highest values from PU. To achieve the second objective, a new cycle starts in Steps (13–26), focusing on selecting a controller server with significant power and the optimal preferred utilization rate. This method, primarily based on CPU utilization loads, differs from most conventional approaches. Steps (14–25) involve assessing the preferred utilization loads for each server.

In Steps (15–24), the server that best matches the highest CPU-preferred utilization or falls within a three-tolerance range is designated as the controller server. This optimal preferred utilization is referred to as "best utilization". Ultimately, this process results in a cluster of controller servers, each characterized by multiple instances of "best utilization".

6.2. A3C algorithm

The primary aim of Algorithm 2 is to identify the most effective VM destinations within the search domain, ensuring that these destinations provide adequate processing power and proper execution while optimizing energy efficiency, CO2 emissions, and reliability. To achieve these objectives, the algorithm employs an enhanced asynchronous advantage actor–critic (A3C) approach, ensuring no compromise on QoS and SLA.

The algorithm processes batch workflow module applications, computing nodes, virtual machines, gradient values, and temporal checkpoint error values. Its objective is to enable reliable workflow scheduling based on the predefined policy $\lambda^{\star}(\alpha_{\tau}|S_p)$, taking into account the failure rates of servers and communication links.

6.2.1. Functionality of A3C steps

In Step (1), the algorithm starts by initializing various parameters, including global network settings and network-specific actor criteria, which are crucial for establishing an effective computational environment. In Steps (3–6), a loop function carries out two main tasks: a layer-based algorithm organizes applications within a workflow into different layers. In contrast, a sort-based topological algorithm prioritizes applications within each layer according to data size and complexity. Another loop function in Steps (7–11) sorts virtual machines on a single server in a data center and assigns priorities based on the server's computational capacity.

Steps (12 and 13) involve initializing gradient values and network parameters to support parallel and periodic updates to the global network. In Step (14), the algorithm sets the values for state and action spaces. Step (15) entails calculating the predefined policy $\lambda(\alpha_{\tau}|S_p;\zeta_{\alpha}^{\sim})$ to monitor the reward function, ensuring the proper organization, scheduling, and execution of workflow applications. After computing

Algorithm 2: A3C Algorithm

Input: $\mathcal{W} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$, $\mathcal{V} = \{v_1, v_2, \ldots, v_k\}$,$\mathcal{D} = \{d_1, d_2, \ldots, d_e\}$, $\zeta_\alpha, \zeta_\alpha^{\sim}, \varpi_\alpha, \varpi_\alpha^{\sim}$
Output: *Developing scheduling decision-making for* $\forall\alpha \in \mathcal{W}$ *over set of computing nodes, using* $\lambda^{\star}(\alpha_\tau|\mathcal{S}_p;\zeta_\alpha)$

1 Initialize $\zeta_\alpha, \zeta_\alpha^{\sim}, \varpi_\alpha, \varpi_\alpha^{\sim}$.
2 $\alpha \leftarrow 1$.
3 **for** $\left(\forall\alpha \in \mathcal{W}\right)$ **do**
4 Conduct layer-based algorithm to classify applications within a workflow into different layers.
5 Conduct sort-based topological algorithm to sort applications within the same layer and assign priority based on the application's data size and complexity.
6 **end**
7 **for** $\left(\forall s \in \mathcal{S}\right)$ **do**
8 **for** $\left(\forall \mathcal{V} \in \mathcal{S}\right)$ **do**
9 Conduct sort-based topological algorithm to sort virtual machine within a single server in data center and assign priority based on the total server's computational capacity.
10 **end**
11 **end**
12 Initialize gradient value and assign: $d\zeta_\alpha \leftarrow 0$ and $\varpi_\alpha \leftarrow 0$.
13 Initialize network parameters as follows: $\zeta_\alpha^{\sim} \leftarrow \zeta_\alpha$ and $\varpi_\alpha^{\sim} \leftarrow \varpi_\alpha$. These parameters are essential for parallel processing and for periodically updating a global network.
14 Feed both $\mathcal{S}_p$ and α_s with necessary input data.
15 Compute the predefined policy $\lambda(\alpha_\tau|\mathcal{S}_p;\zeta_\alpha^{\sim})$ so that the paradigm can arrange, schedule, and maintain workflow applications.
16 Compute the rewarding function using Equation (4) and get the desired value. This is a continual process to keep reward function updated.
17 Jump to the next state space, $\mathcal{S}_p$.
18 Increase the global shared counter and the step counter.
19 Stop the loop counter when $\forall\alpha \in \mathcal{W}$ being assigned and placed on their desired destinations.
20 Assess the value function, $\mathcal{V}^{\lambda}(\mathcal{S}_p)$ using Equation (6).
21 **for** $\left(\forall\alpha \in \mathcal{W}\right)$ **do**
22 Compute the value function, $\mathcal{V}^{\lambda}(\mathcal{S}_p)$ using Equation (6) to expect the outcomes of the state and action sequence determined by the action and state spaces.
23 Compute the gradient value $d\zeta_\alpha$ using Equation (10) to help the system optimize converging biases during the solution selection process.
24 Compute temporal check point error $d\varpi_\alpha$ using Equation (12) to handle the update calculation for each agent from $\mathcal{S}_p$ to $\mathcal{S}_{p+1}$.
25 Evaluate the following parameters: energy consumption rate, carbon emissions costs, and scheduling reliability using Equations (20), (23), and (25).
26 **end**
27 Update the values of the followings: ζ_α by $d\zeta_\alpha$ and ϖ_α by $d\varpi_\alpha$.
28 Continue until all applications within a workflow are being scheduled and send back $\lambda^{\star}(\alpha_\tau|\mathcal{S}_p;\zeta_\alpha)$.

the reward function in Step (16) using Eq. (4), the algorithm checks whether the parameter values align with the expected outcomes from the training phase. If the scheduling decision meets the expectations, it updates the values as optimized, adjusting global and local network parameters. If not, it calculates the accumulated gradient value to suggest improved scheduling decisions for the policy function in Step (17). This process continues until all iterations are completed in Step (19).

The final loop function in Steps (21–26) performs several operations: it calculates the value function $\mathcal{V}^{\lambda}(\mathcal{S}_p)$ using Eq. (6) to predict the outcomes of the state and action sequence determined by the action and state spaces; it computes the gradient value $d\zeta_\alpha$ using Eq. (10) to optimize converging biases during solution selection; it calculates the temporal checkpoint error $d\varpi_\alpha$ using Eq. (12) to manage update calculations for each agent from $\mathcal{S}_p$ to $\mathcal{S}_{p+1}$; and it evaluates parameters such as energy consumption rate, carbon emissions costs, and scheduling reliability using Eqs. (20), (23), and (25). The algorithm updates ζ_α with $d\zeta_\alpha$ and ϖ_α with $d\varpi_\alpha$.

6.3. SLA-QoS-RALOC algorithm

The main goal of Algorithm 3 and 4 are to handle resource allocation in a way that is aware of SLA and QoS constraints, using the SLA-QoS-Aware Resource Allocation (SLA-QoS-RALOC) algorithm to optimize workflow applications.

After computing resources are allocated, two types of applications require continuous monitoring: CPU-bound and memory-bound. CPU-bound applications are those that perform complex computations and rely heavily on CPU resources. When considering VM migration, the VM with the highest CPU usage is typically selected. Among these high-CPU VMs, priority is given to those with the most significant memory usage. However, migrating VMs with high memory demands can be resource-intensive, especially if the processing node has limited memory capacity. This can slow down migrations, potentially causing delays and insufficient memory for new tasks, which may affect system performance. Nevertheless, successfully migrating high-memory VMs can be advantageous as it frees up memory on the host machine, allowing it to handle new requests and enhance overall system efficiency.

To maintain computing resources in the data center, two main operations are performed: the first is to define a dynamic, adaptable threshold to identify overloaded and underloaded computing nodes, and the second is VM selection and allocation, which involves choosing suitable computing nodes and distributing them efficiently across available VMs to optimize performance and resource utilization.

6.3.1. Dynamic adaptive thresholds (DAT)

The thresholds in the A3C-CGT framework are adjustable based on the characteristics of the computing environment. Typically, these thresholds are set using the *iQr* approach, which establishes threshold values for each group by segmenting the dataset into quartiles. This quartile-based approach makes the thresholds adaptable to the existing conditions.

While putting these values, it is crucial to compute the interquartile range. This process arranges information in ascending order, and the median is determined. This is followed by partitioning the data set into two halves and determining the median for each half. Eventually, it becomes easy to compute the interquartile range as a difference between the upper and lower quartile values, showing the spread in the middle 50% of a data set.

$$iQr = Q_u - Q_l \tag{38}$$

The upper quartile is Q_u, and the lower quartile is Q_l. There are high and low threshold values, denoted as τ_u and τ_l, respectively, in the A3C-CGT algorithm. The low threshold value is found by calculating the median value of the first half of the ordered dataset. In contrast, the upper threshold is calculated from the median of the second half. When a computing device monitors its CPU or memory usage, it compares the utilization levels to the thresholds specific to the data center. If the utilization level exceeds the upper threshold (τ_u), the A3C-CGT algorithm identifies the server as overloaded, necessitating VM migrations from that host. Conversely, if the utilization level is below the lower threshold (τ_l), the server is considered underloaded, and VMs can be migrated to other hosts.

The workload's variability can cause fluctuations in the utilization of the current processing server over time. Consequently, CPU utilization may exceed the upper threshold if VMs are not migrated promptly. The detection algorithm for identifying overutilization evaluates the server's CPU and memory usage.

6.3.2. Functionality of SLA-QoS-RALOC: DAT steps

Algorithm 3 is designed to detect overloaded servers through a continuous loop encompassing Steps (2–32). This loop ensures constant monitoring of CPU and memory usage across servers to identify instances of overloading.

In Steps (3–4), the algorithm employs two practical functions to obtain the current server's CPU and memory utilization. It then checks these values against the highest threshold. The server is marked as an overcrowded node if the CPU and memory utilization exceeds this threshold (Steps 5–16).

Similarly, in Steps (8–9), the algorithm gathers information on all VMs' CPU and memory consumption. Steps (10–14) involve a loop that identifies VMs responsible for server overcrowding that exceeds the server's upper threshold. The overloaded VMs are then stored in a new array. By continuously monitoring CPU and memory usage and detecting instances of overloading, the algorithm helps prevent server crashes and enhances the overall system performance.

Most existing resource allocation frameworks use a straightforward approach to identify underutilized nodes, typically relying on heuristics that select VMs with minimum CPU utilization for migration. However, our paradigm involves heterogeneous underlying cloud network placement models and uses an overloaded node detection algorithm that considers CPU and memory usage. Therefore, we propose a novel, underloaded node detection model considering the node's CPU and memory consumption. The detection algorithm for underloaded nodes differs from the one used for detecting overloaded nodes and presents two potential scenarios. In the first scenario (a), underloaded nodes may be transformed into active nodes by migrating VMs from other underloaded nodes. In the second scenario (b), underloaded nodes may be shut down to save energy.

To detect underloaded processing servers, Algorithm 3 utilizes Steps (17–28). During these steps, the algorithm determines all servers' current CPU and memory usage and compares this data to predefined lower thresholds. If a server is below these thresholds or has been shut down, it is marked as underloaded and added to a new array. Additionally, the algorithm evaluates whether the loaded servers identified can host additional VMs and return to normal operation or if all VMs need to be migrated to shut down the server and conserve energy. To achieve this, the algorithm examines the CPU and memory consumption of executable VMs, comparing the data against a predefined threshold, as demonstrated in Steps (22–26). The algorithm also identifies underloaded VMs and selects them for migration, as outlined in Steps (23–25).

6.3.3. VM selection and allocation (VMSαA)

Algorithm 4 for VM selection maintains server utilization within a normal operating range by considering the PPR processing servers. It relocates VMs with the lowest CPU utilization to reduce the risk of a significant increase in utilization, thereby preventing QoS violations. The algorithm aims to identify the most efficient destination for VMs that have migrated from OVS or UNS. It ensures that VMs are allocated to nodes with enough processing capacity, optimal QoS, and energy efficiency. The method involves feeding an array of migrated VMs to get their best destinations.

6.3.4. Functionality of SLA-QoS-RALOC: VMSαA steps

Step (1) involves initializing parameters, including the total number of migrated virtual machines and the number of iterations. Step (2) sets a range for the inertia weight coefficient, specifying minimum and maximum values. Step (3) defines the minimum and maximum capacities for virtual machines. In Step (4), a window matrix is created for each computing node, allowing the scheduler to search for and allocate available time slots for all virtual machines.

Steps (6–10) include calculating the current CPU and memory usage of the migrated virtual machines using the window matrix, which helps identify available time slots on allocated servers for efficient resource allocation. Step (8) applies a sorting algorithm to rank virtual machines in decreasing order of priority. The sorted virtual machines are then stored in a new array named $\mathcal{SV}$ in Step (9). The algorithm begins with the first allocated server, examines its window matrix to find available time slots in Step (13), and verifies the presence of these time slots in Step (14). If available, Step (15) compares the virtual machine's required capacity with the server's capacity from the first time slot onward.

In Step (16), the algorithm assigns the migrated virtual machine if the capacities match; otherwise, it proceeds to the next available server. This process continues until all migrated virtual machines from OVN and UNS are allocated in Step (17).

7. Experimental analysis

This section first outlines the simulation's experimental environment and setup, then presents the findings related to synthesis and real-world workflow applications.

7.1. Experimental environment

The CloudSim simulation is carried out to develop an effective cloud network placement, allowing for dynamic resource allocation and management as required. Table 3 details the simulation configuration components used in our work.

Algorithm 3: SLA-QoS-RALOC: Dynamic Adaptive Thresholds

Input: $\mathcal{W} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}, \mathcal{S} = \{s_1, s_2, \ldots, s_m\}, \mathcal{V} = \{v_1, v_2, \ldots, v_k\}, \mathcal{D} = \{d_1, d_2, \ldots, d_e\}, \tau_u, \tau_l$
Output: *Improve the performance of computing nodes in a data center that is either overburdened or underutilized.*

Identify such computing nodes in the cloud network that offer the best performance-to-power ratio.
for $\left(\forall s \in \mathcal{S}\right)$ **do**
  Calculate the actual CPU usage of the compute nodes and save this in a new array called α_{cpu}. This way, before adding new applications to the nodes, the algorithm could look into cpu allocation for the nodes.
  Calculate the current memory allocation for the computing nodes and store the results in a new array called α_{mem}. The algorithm will thus be able to check how much memory is allocated to each node before making any new allocation.
  if $\left((\alpha_{cpu} \geq \tau_h) \vee (\alpha_{mem} \geq \tau_h)\right)$ **then**
    Define a new array $\mathcal{OVN}$, and from it, select the computing nodes whose current CPU utilization and memory usage surpass the maximum thresholds.
    for $\left(\forall v \in \mathcal{V}\right)$ **do**
      Compute the CPU utilization from the allocated VMs to calculate and add the values into a new array named δ_{cpu}. Now, the algorithm can identify which virtual machines have high CPU utilization.
      Calculate the current memory usage of the allocated VMs and save the results in a new array called δ_{mem}. This will help the algorithm identify which VMs consume high amounts of memory.
      if $\left((\delta_{cpu} \geq \tau_h) \vee (\delta_{mem} \geq \tau_h)\right)$ **then**
        Define a new array to explain how the Virtual Machines overutilize the Computing Node's CPU Power and Memory Utilization.
        Select the underutilized virtual machines and candidates for migration to another computing node.
        Combine the migrated Virtual Machines with preallocated Virtual Machines to avoid overlap and window matrix conflict.
      end
    end
  end
  else
    Define a new array UNS whose elements are computing nodes currently under the lower limits for CPU and memory use.
    for $\left(\forall v \in \mathcal{V}\right)$ **do**
      Compute the current CPU utilization of the allocated virtual machines and save it into a new array η_{cpu}. This will enable the algorithm to detect virtual machines that lightly utilize the CPU.
      Calculate the current memory consumption of the assigned VMs and write to a new array η_{mem}—this way, the algorithm can catch VMs that use memory negligibly small.
      if $\left((\eta_{cpu} \geq \tau_h) \vee (\eta_{mem} \geq \tau_h)\right)$ **then**
        Define an array by finding the virtual machines that are least utilized by the computing node in terms of power capacity and memory.
        Choose those virtual machines that can be transferred from one computing node to another.
        Add up the previous calculations of the virtual machines that have migrated with the already preallocated ones so as not to overlap or conflict with the window matrix.
      end
    end
  end
end

7.1.1. Application graph settings

Creating the synthetic datasets involved modifying the structure and attributes of the application graph. These datasets included applications characterized by the complexity and number of charges and the volume and size of data transferred during communications between applications. Additionally, we implemented our algorithm using real scientific workflow applications, specifically Cybershake and Montage, to evaluate its efficiency. The datasets are categorized based on the nature of incoming user requests, ranging from small (50 to 90 tasks) to extra-large (220 to 500 tasks), with intermediate categories like small-medium (100 to 120 tasks), medium (130 to 175 tasks), and large (180 to 210 tasks). This analysis aimed to assess how the system handles various application volumes and identify potential improvement areas.

7.1.2. Network graph settings

The underlying cloud network consists of various servers, including different instance types. The allocated servers have a capacity ranging from 1860 to 2860 MIPS and are equipped with a Core i5 CPU and 8 GB of RAM. Different VM types are installed based on CPU usage, available on-demand, and can be reused for new requests to minimize startup, idle, and shutdown overheads. Additionally, our network graph includes components such as the number of nodes, processing capabilities, failure rates, the number of links, their bandwidth, failure rates, and minimum delay per link.

7.2. Experimental results

The proposed A3C-CGT paradigm is compared to task-scheduling methods like MSISCSOA [14], SDAR-NFAT [15], DCOHHOTS [16], A2C [17], DQN [18], and EBABC-PF [19]. The practical analysis involved two scheduling experiments to assess performance using various metrics: the first focused on scheduling synthesized workflows, while the second dealt with real-world workflows.

Algorithm 4: SLA-QoS-RALOC: VM Selection and Allocation

Input: $\mathcal{W} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}, \mathcal{S} = \{s_1, s_2, \ldots, s_m\}, \mathcal{V} = \{v_1, v_2, \ldots, v_k\}, \mathcal{D} = \{d_1, d_2, \ldots, d_e\}, \mathcal{OVN}, UNS$
Output: *Efficiently locate the optimal virtual machine for hosting workflow applications using advanced window matrix techniques.*

Initialize parameters such as the total number of migrated virtual machines and the number of iterations.
Specify the range of the inertia weight coefficient (minimum and maximum values).
Set the minimum and maximum capacities for virtual machines.
Define a window matrix for each computing node to allow the scheduler to search and allocate available time slots for all virtual machines.
$\mathcal{Y}_\tau \leftarrow$MAXVALUE

for $\left(\forall s \in (\mathcal{OVN}) \vee (UNS)\right)$ **do**
    Calculate migrated virtual machines' current CPU and memory usage using a window matrix. This approach will help identify available time slots in allocated servers, facilitating efficient resource allocation.
    Implement a sorting algorithm to prioritize virtual machines in decreasing order of their importance.
    Store the sorted virtual machines in a new array named $\mathcal{SV}$.
end
for $(\forall s \in \mathcal{S})$ **do**
    for $(\forall \mathcal{V} \in \mathcal{SV})$ **do**
        Begin with the first allocated server and examine its window matrix to identify any available time slots.
        If available, compare the virtual machine's required capacity with the server's capacity starting from the first time slot through to the end.
        Assign the migrated virtual machine if both capacities align; otherwise, proceed to the next available server.
        Repeat this process until all migrated virtual machines from OVN and UNS are consolidated.
    end
end

Table 3
Simulation settings: extensive configuration.

Assessment metric	Numerical values
A3C-CGT model simulation setup	
Simulation procedure duration	1200 s
Iteration EED length	0.237–0.431 s
Demands assigning rate	20–85
Processing service rate	0.512–0.934 s
Operating System (OS) configuration	
Time-sensitive service initiation	≈213 s
Operating System provider	Windows education 11 with [x86]-32 bit capacity
Simulation environment	Java language
EED duration	≈43 s
Computing capability	Intel® Five Core™
Memory allocation capability	8 GB
Workflow application settings	
Incoming requests have priority	≈0.15–0.28 megabyte
Peak time for service delivery	≈0.375–0.937 s
Outgoing demand priority	≈0.23–0.45 megabyte
Memory assignment	80–412 megabyte
Number of module demands	25–63 TI
Module application scheduling	100–800 applications
Servers and virtual machine configuration	
Complete virtual machines	800 virtual machines
Time of network usage	≈0.028–0.047 $
Resource service budge	≈0.025–0.067$
Capability of virtual machine	2.5 GB
VM EED termination	88 s
VM termination duration	13 s
Computing unit costs	≈0.15–0.25$
Ability of communication link	25–1024 Mbps
Ability of computing processing	≈1860–2860 MIPS

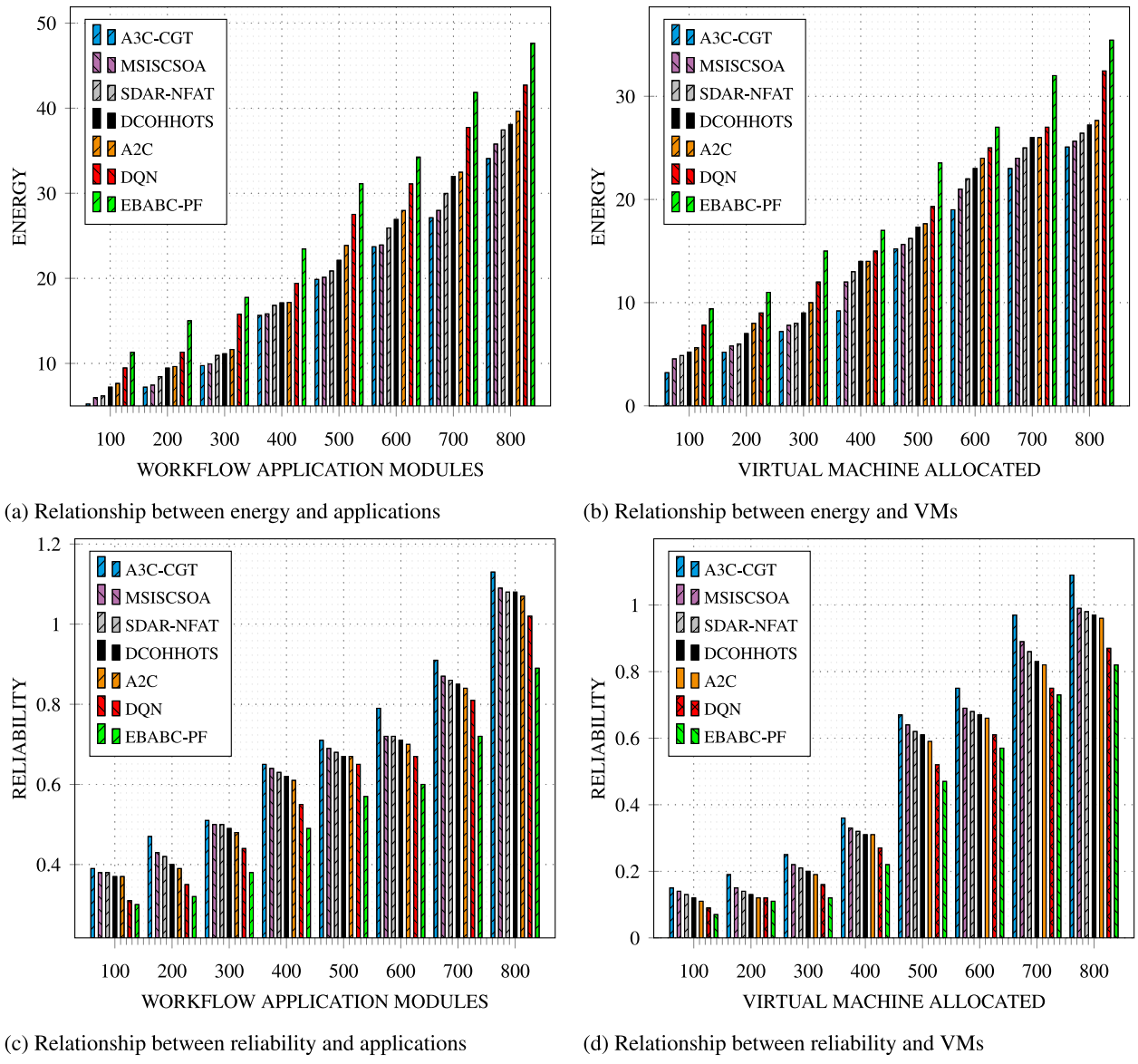
7.2.1. Paradigm comparison under synthesized datasets

Fig. 5(a) shows how the A3C-CGT paradigm stacks up against algorithms MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF, with a focus on how much energy they use. The comparison includes two analyses: one assessing performance under various workloads and another considering different numbers of VMs. Our A3C-CGT method demonstrated superior energy efficiency compared to the existing algorithms. For example, with a workload of 100, the energy consumption figures for the algorithms were 5.23, 5.98, 6.18, 7.21, 7.65, 9.47, and 11.30, respectively. When the workload size increases to 500, the energy consumption for A3C-CGT is approximately 19.87, while for MSISCSOA, it is about 20.12; for SDAR-NFAT, roughly 20.87; for DCOHHOTS, 22.12; for A2C, 23.87; for DQN, 27.52; and for EBABC-PF, 31.12. When the workload increased to 800, A3C-CGT consumed approximately 34.08, MSISCSOA about 35.78, SDAR-NFAT roughly 37.44, DCOHHOTS 38.08, A2C 39.64, DQN 42.71, and EBABC-PF 47.64.

Fig. 5(b) further compares our algorithm against MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF regarding the number of VMs, focusing on energy costs. With 100, the energy rate for our paradigm is 3.21, while for MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF, the costs escalate to 4.55, 4.87, 5.21, 5.64, 7.82, and 9.41, respectively. When VMs reach 500, the energy costs for our paradigm are 15.21, for MSISCSOA 15.64, for SDAR-NFAT 16.22, for DCOHHOTS 17.32, for A2C 17.65, for DQN 19.32, and for EBABC-PF 23.54. At 800, the energy values are as follows: A3C-CGT 25.08, MSISCSOA 25.64, SDAR-NFAT 26.42, DCOHHOTS 27.21, A2C 27.65, DQN 32.45, and EBABC-PF 35.44.

Our paradigm surpasses MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF by using the DVFS technique, which controls CPU frequencies to ensure each VM runs efficiently. Integrating a collaborative training model enhances our approach, forming efficient server coalitions for an optimized performance-to-power ratio.

Fig. 5(c) shows how the A3C-CGT paradigm stacks up against MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF in terms of failure rate metrics that show how reliable the server and communication link are. The comparison includes two tests: one with different workloads and another with varying numbers of VMs. Our A3C-CGT method has shown significant reliability improvements compared to MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF, especially in failure rates due to the number of applications. For instance, with 100 applications, the reliability figure for A3C-CGT was 0.39, for MSISCSOA 0.38, for SDAR-NFAT 0.38, for DCOHHOTS 0.37, for A2C 0.37, for DQN 0.31, and for EBABC-PF 0.30. When the number of tasks increases to 500, the reliability values are A3C-CGT 0.71, MSISCSOA 0.69, SDAR-NFAT 0.68, DCOHHOTS 0.67, A2C 0.67, DQN 0.65, and EBABC-PF 0.57. With 800 applications, the reliability values are A3C-CGT 1.13, MSISCSOA 1.09, SDAR-NFAT 1.08, DCOHHOTS 1.08, A2C 1.07, DQN 1.02, and EBABC-PF 0.89.



(a) Relationship between energy and applications

(b) Relationship between energy and VMs

(c) Relationship between reliability and applications

(d) Relationship between reliability and VMs

Fig. 5. The effect of energy and reliability on the quantity of workflow module applications and virtual machines.

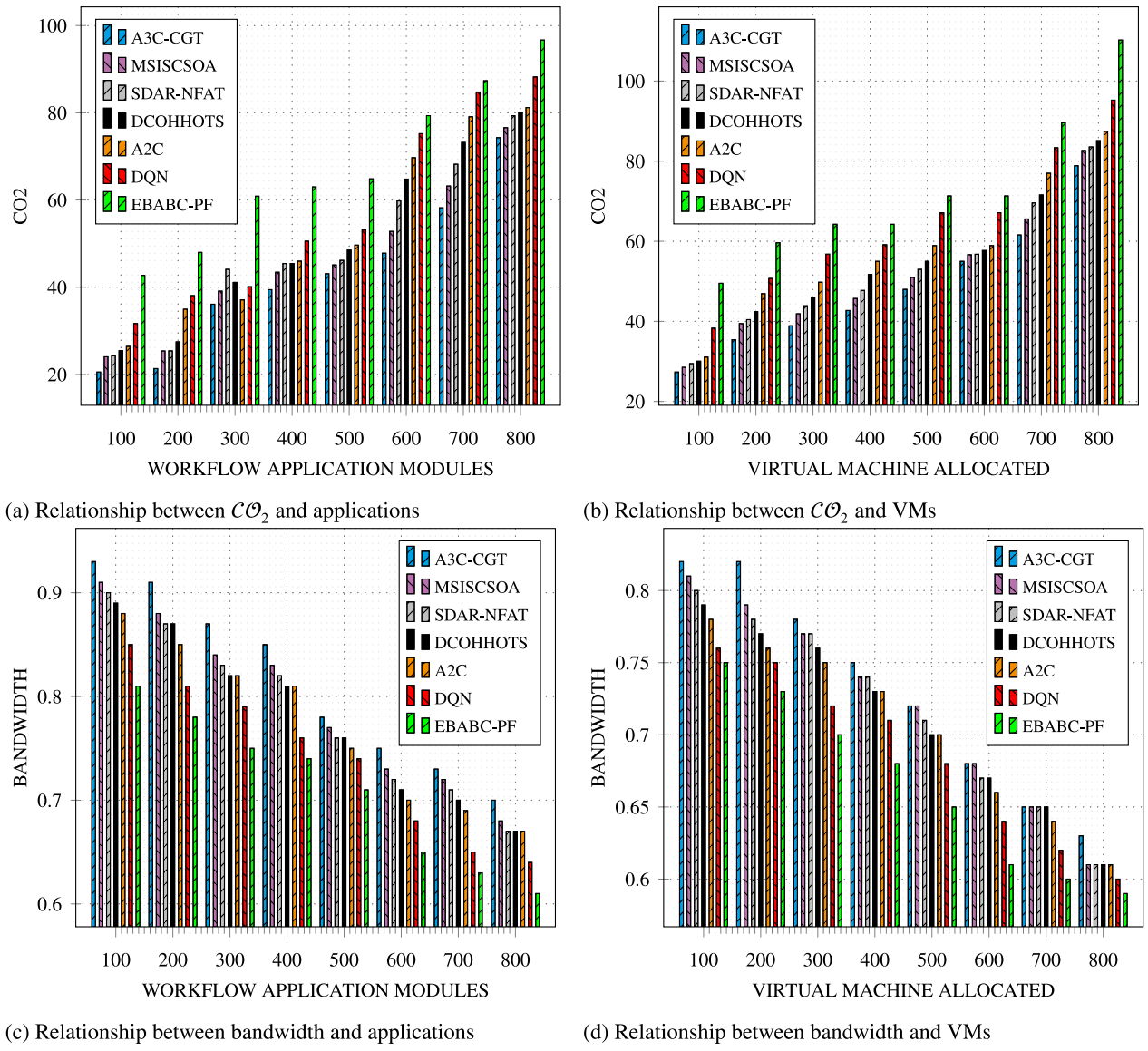
Fig. 5(d) contrasts our model regarding VM reliability with MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF. With 100 VMs, the reliability of our paradigm was 0.15, compared to MSISCSOA 0.14, SDAR-NFAT 0.13, DCOHHOTS 0.12, A2C 0.11, DQN 0.09, and EBABC-PF 0.07. With 500 VMs, the reliability values are A3C-CGT 0.67, MSISCSOA 0.64, SDAR-NFAT 0.62, DCOHHOTS 0.61, A2C 0.59, DQN 0.52, and EBABC-PF 0.47. With 800 VMs, the reliability values are A3C-CGT 1.09, MSISCSOA 0.99, SDAR-NFAT 0.98, DCOHHOTS 0.97, A2C 0.96, DQN 0.87, and EBABC-PF 0.82.

Our paradigm's main advantage over MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF lies in its reliability model, which assesses server and communication link failure probabilities, significantly enhancing system dependability.

Our proposed paradigm outperforms current algorithms in various scenarios, as illustrated in Fig. 6(a), focusing on reducing carbon dioxide emissions. The new approach achieves this by optimizing server frequencies and VM allocation and employing the VM reuse method. Initially, all algorithms exhibit low CO_2 emissions. However, as the number of applications increases, emissions also rise. For instance, with 100 applications, A3C-CGT emits 20.55 units of CO_2, compared to MSISCSOA 24.05, SDAR-NFAT 24.27, DCOHHOTS 25.45, A2C 26.43, DQN 31.66, and EBABC-PF 42.71. These figures remain consistent even with 500 applications: A3C-CGT 43.09, MSISCSOA 45.11, SDAR-NFAT 46.16, DCOHHOTS 48.52, A2C 49.65, DQN 53.14, and EBABC-PF 64.85 units. With 800 applications, the emissions are A3C-CGT 74.29, MSISCSOA 76.58, SDAR-NFAT 79.32, DCOHHOTS 80.05, A2C 81.20, DQN 88.22, and EBABC-PF 96.67 units.

Fig. 6(b) compares our paradigm with MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF regarding VM usage and CO_2 emissions. Initially, our method significantly reduces CO_2 emissions, emitting 27.34 units for 100 VMs, compared to MSISCSOA 28.52, SDAR-NFAT 29.44, DCOHHOTS 30.05, A2C 31.07, DQN 38.30, and EBABC-PF 49.50. With 500 application tasks, the emissions are A3C-CGT 54.98, MSISCSOA 56.64, SDAR-NFAT 56.75, DCOHHOTS 57.69, A2C 58.90, DQN 67.12, and EBABC-PF 71.32. With 800 VMs, the emissions are A3C-CGT 78.87, MSISCSOA 82.66, SDAR-NFAT 83.55, DCOHHOTS 85.12, A2C 87.44, DQN 95.23, and EBABC-PF 110.23.

The superiority of our algorithm comes from three key improvements: utilizing the DVFS technique for CPU frequency adjustment, optimizing energy consumption percentages, and avoiding older servers and communication links prone to high failure rates.



(a) Relationship between $\mathcal{CO}_2$ and applications

(b) Relationship between $\mathcal{CO}_2$ and VMs

(c) Relationship between bandwidth and applications

(d) Relationship between bandwidth and VMs

Fig. 6. The effect of $\mathcal{CO}_2$ and bandwidth dependency on the quantity of workflow module applications and virtual machines.

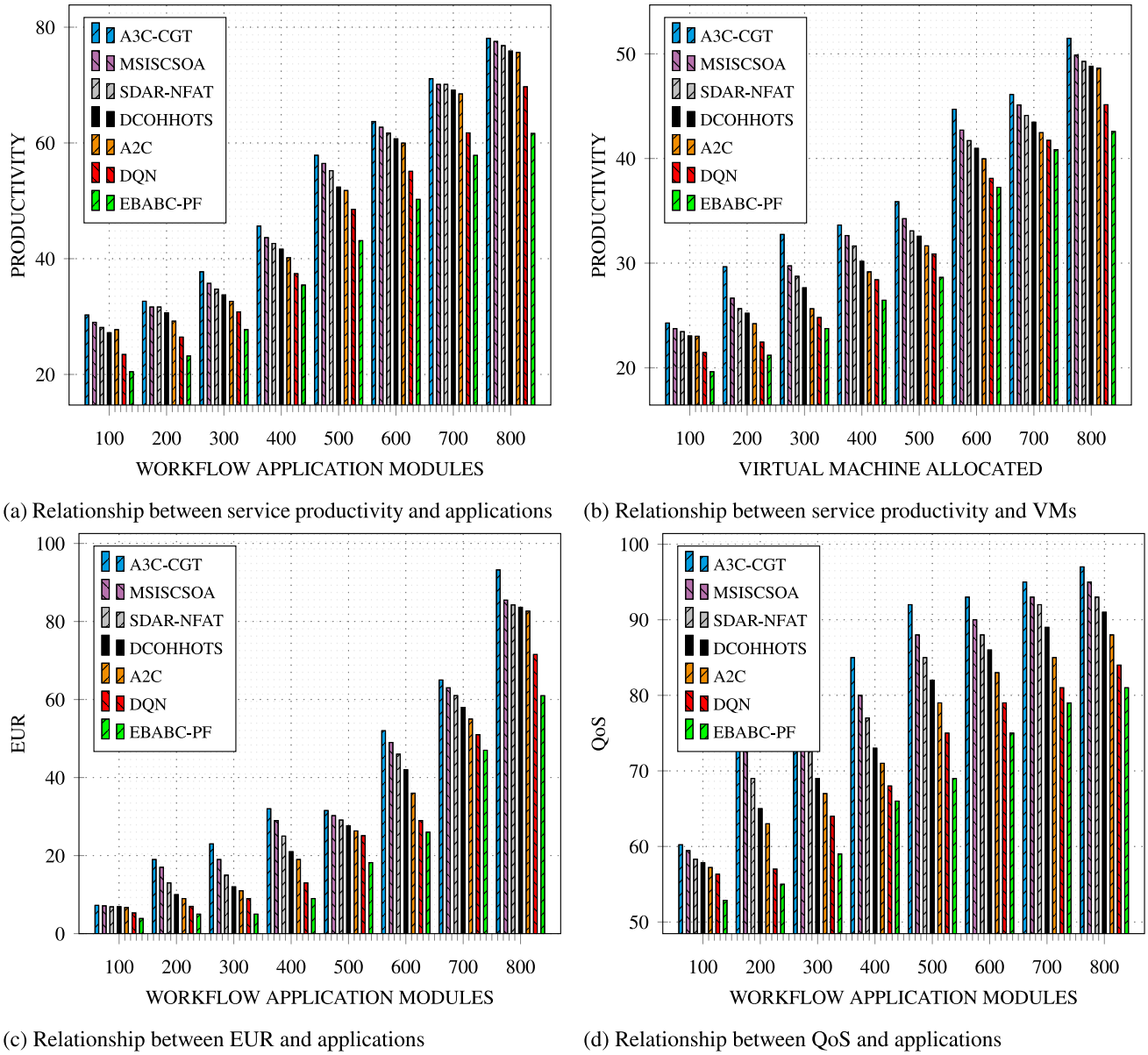
Fig. 6(c) shows A3C-CGT's performance compared to MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF in terms of bandwidth dependency on the number of applications. Initially, with 100 applications, A3C-CGT improved bandwidth dependency to 0.93, surpassing MSISCSOA's 0.91, SDAR-NFAT's 0.90, DCOHHOTS's 0.89, A2C's 0.88, DQN's 0.85, and EBABC-PF's 0.81. With 500 applications, A3C-CGT maintained a bandwidth dependency improvement of 0.78, while MSISCSOA got 0.77, SDAR-NFAT 0.76, DCOHHOTS 0.76, A2C 0.75, DQN 0.74, and EBABC-PF 0.71. With 800 applications, the values are A3C-CGT 0.70, MSISCSOA 0.68, SDAR-NFAT 0.67, DCOHHOTS 0.67, A2C 0.67, DQN 0.64, and EBABC-PF 0.61.

Fig. 6(d) shows that A3C-CGT is more effective regarding the number of VMs and how they affect bandwidth dependency. For 100 VMs, A3C-CGT consistently performs better than the others, with an improvement rate of 0.82 compared to 0.81 for MSISCSOA, 0.80 for SDAR-NFAT, 0.79 for DCOHHOTS, 0.78 for A2C, 0.76 for DQN, and 0.75 for EBABC-PF. With 500 modules, the bandwidth dependency values are A3C-CGT 0.72, MSISCSOA 0.72, SDAR-NFAT 0.71, DCOHHOTS 0.70, A2C 0.70, DQN 0.68, and EBABC-PF 0.65. With 800 modules, the values are A3C-CGT 0.63, MSISCSOA 0.62, SDAR-NFAT 0.61, DCOHHOTS 0.61, A2C 0.61, DQN 0.60, and EBABC-PF 0.59.

A3C-CGT's superior bandwidth management is due to its effective use of bandwidth channels and consideration of bandwidth link failure rates when tasks are sent to cloud servers.

Fig. 7(a) compares A3C-CGT with MSISCSOA, SDAR-NFAT, DCOHHOTS, DQN, A2C, and EBABC-PF in terms of application count and service productivity. At 100 applications, A3C-CGT achieved the highest service productivity with a score of 30.27, followed by MSISCSOA with 29.01, SDAR-NFAT with 28.12, DCOHHOTS with 27.21, A2C with 27.74, DQN with 23.45, and EBABC-PF with 20.47. With 500 modules, A3C-CGT continued to lead with a score of 57.87, while MSISCSOA scored 56.45, SDAR-NFAT scored 55.21, DCOHHOTS scored 52.37, A2C scored 51.78, DQN scored 48.52, and EBABC-PF scored 43.12. At 800 modules, A3C-CGT maintained a score of 78.08, with MSISCSOA at 77.54, SDAR-NFAT at 76.82, DCOHHOTS at 75.88, A2C at 75.64, DQN at 69.71, and EBABC-PF at 61.64.

Fig. 7(b) evaluates A3C-CGT's performance against these algorithms based on VM count and service productivity efficiency. A3C-CGT consistently performed better. At 100 applications, A3C-CGT had a service productivity score of 24.27, compared to MSISCSOA's 23.74, SDAR-NFAT's 23.45, DCOHHOTS's 23.04, A2C's 23.01, DQN's 21.45, and EBABC-PF's 19.62. With 500 modules, A3C-CGT scored 35.87, MSISCSOA scored 34.25, SDAR-NFAT scored 33.08, DCOHHOTS scored



(a) Relationship between service productivity and applications

(b) Relationship between service productivity and VMs

(c) Relationship between EUR and applications

(d) Relationship between QoS and applications

Fig. 7. The effect of service productivity, EUR, and QoS on the quantity of workflow module applications and virtual machines.

32.57, A2C scored 31.65, DQN scored 30.88, and EBABC-PF scored 28.64. At 800 modules, A3C-CGT maintained a score of 51.49, followed by MSISCSOA with 49.88, SDAR-NFAT with 49.28, DCOHHOTS with 48.82, A2C with 48.64, DQN with 45.14, and EBABC-PF with 42.58.

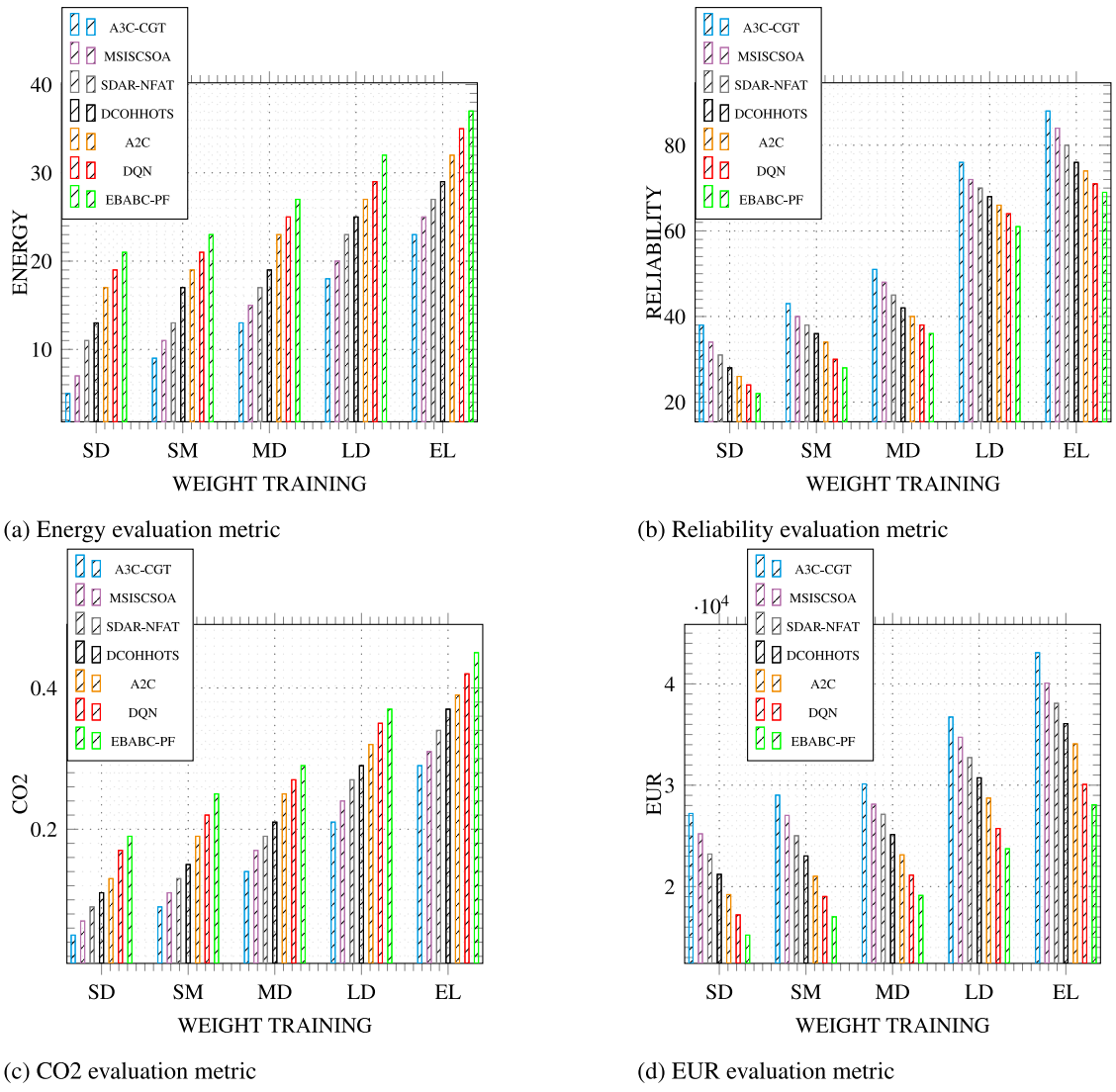
A3C-CGT's superior service productivity utilization is due to its strategic selection of application-VM combinations, optimizing the VM matrix window for maximum efficiency.

Fig. 7(c) shows the performance evaluation of several algorithms – A3C-CGT, MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF – based on their effectiveness in multiple applications using service measurement metrics. With 100 applications, A3C-CGT scored 7.25, MSISCSOA scored 7.13, SDAR-NFAT scored 6.88, DCOHHOTS scored 6.91, A2C scored 6.71, DQN scored 5.31, and EBABC-PF scored 3.89. When the number of applications increased to 500, A3C-CGT scored 31.55, MSISCSOA scored 30.24, SDAR-NFAT scored 29.11, DCOHHOTS scored 27.66, A2C scored 26.33, DQN scored 25.11, and EBABC-PF scored 18.22. For 800 applications, A3C-CGT scored 93.22, MSISCSOA scored 85.47, SDAR-NFAT scored 84.25, DCOHHOTS scored 83.64, A2C scored 82.66, DQN scored 71.54, and EBABC-PF scored 60.98.

Fig. 7(d) compares the A3C-CGT algorithm with MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF regarding QoS for application ranges of 100–800. With 100 applications, A3C-CGT scored 60.22, MSISCSOA scored 59.45, SDAR-NFAT scored 58.31, DCOHHOTS scored 57.84, A2C scored 57.21, DQN scored 56.33, and EBABC-PF scored 52.87. For 500 applications, A3C-CGT scored 92, MSISCSOA scored 88, SDAR-NFAT scored 85, DCOHHOTS scored 82, A2C scored 79, DQN scored 75, and EBABC-PF scored 69. For 800 applications, A3C-CGT scored 97, MSISCSOA scored 95, SDAR-NFAT scored 93, DCOHHOTS scored 91, A2C scored 88, DQN scored 84, and EBABC-PF scored 81.

7.2.2. Paradigm comparison under real-world datasets

For energy consumption, Fig. 8(a) illustrates the performance of the A3C-CGT algorithm compared to MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF. The data shows that A3C-CGT is more efficient regarding energy usage across various application volumes. For example, A3C-CGT's energy consumption at smaller workload sizes is 5, whereas MSISCSOA is at 7, SDAR-NFAT is at 11,



(a) Energy evaluation metric

(b) Reliability evaluation metric

(c) CO2 evaluation metric

(d) EUR evaluation metric

Fig. 8. CYBERSHAKE: energy, reliability, CO_2, and EUR evaluation metrics.

DCOHHOTS is at 13, A2C is at 17, DQN is at 19, and EBABC-PF is at 21. As workload sizes increase significantly, the energy consumption of the algorithms scales accordingly: A3C-CGT at 23, MSISCSOA at 25, SDAR-NFAT at 27, alg at 29, A2C at 32, DQN at 35, and EBABC-PF at 37.

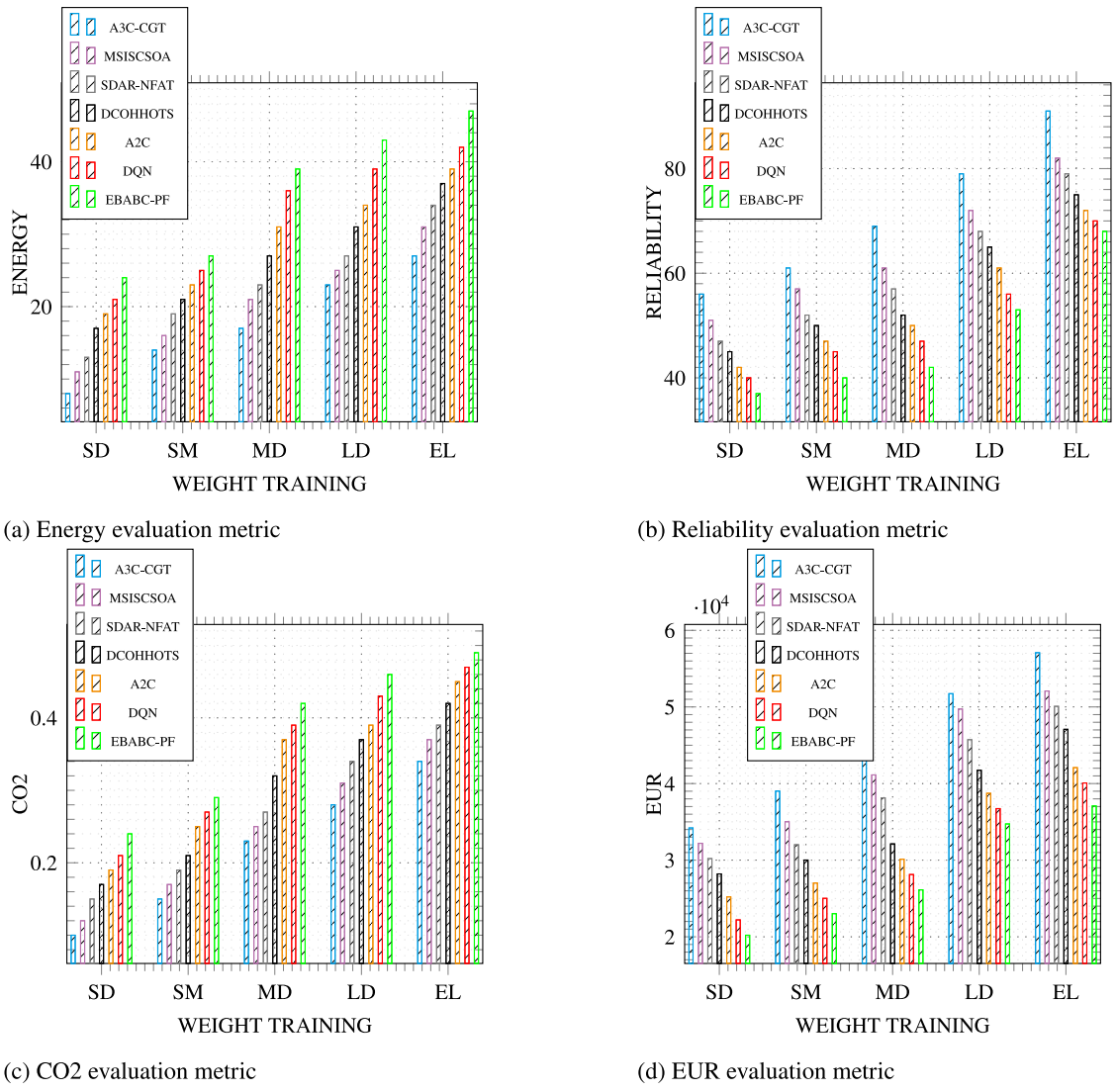
Fig. 8(b) compares the reliability of the A3C-CGT algorithm with MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF. It shows that as the number of applications increases, the reliability of all algorithms decreases. However, A3C-CGT consistently maintains higher server and link reliability efficiency across different application volumes. Specifically, A3C-CGT achieves a reliability rate of 38 at lower application counts, while MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF score 34, 31, 28, 26, 24, and 22, respectively. At higher application counts, A3C-CGT achieves reliability ratings of 88, compared to MSISCSOA at 84, SDAR-NFAT at 80, DCOHHOTS at 76, A2C at 74, DQN at 71, and EBABC-PF at 64.

Fig. 8(c) examines carbon dioxide (CO2) emissions, which are closely related to energy consumption. The figure illustrates that A3C-CGT emits less CO2 than MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF. For instance, A3C-CGT emits 0.05 units of CO2 at smaller workload levels, whereas MSISCSOA consumes 0.07, SDAR-NFAT emits 0.09, DCOHHOTS consumes 0.11, A2C emits 0.13, DQN emits 0.17, and EBABC-PF emits 0.19. At higher workload levels, these emissions increase: A3C-CGT emits 0.29, MSISCSOA consumes 0.31, SDAR-NFAT emits 0.034, DCOHHOTS consumes 0.37, A2C emits 0.39, DQN emits 0.42, and EBABC-PF emits 0.45 units of CO2, respectively.

Fig. 8(d) compares the A3C-CGT algorithm with MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF based on the EUR metric across different applications. At lower application levels, A3C-CGT demonstrates superior EUR effectiveness compared to MSISCSOA by 121, SDAR-NFAT by 124, DCOHHOTS by 127, A2C by 131, DQN by 187, and EBABC-PF by 217. As application levels increase to extreme levels, A3C-CGT maintains significant improvement margins of 251 over MSISCSOA, 266 over SDAR-NFAT, 268 over DCOHHOTS, 272 over A2C, 280 over DQN, and 410 over EBABC-PF, respectively.

Fig. 9(a) illustrates the comparative energy efficiency of the A3C-CGT algorithm against MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF. The results indicate that A3C-CGT consumes less energy across various application sizes. For instance, at smaller workload levels, A3C-CGT's energy consumption is 8, whereas MSISCSOA consumes 11. SDAR-NFAT consumes 13, DCOHHOTS consumes 17, A2C consumes 19, DQN consumes 21, and EBABC-PF consumes 24. As workload sizes increase significantly, A3C-CGT's energy consumption remains lower at 27, compared to 31 for MSISCSOA, 34 for SDAR-NFAT, 37 for DCOHHOTS, 39 for A2C, 42 for DQN, and 47 for EBABC-PF.

Fig. 9(b) examines the reliability performance of the A3C-CGT algorithm compared to MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN,



(a) Energy evaluation metric

(b) Reliability evaluation metric

(c) CO2 evaluation metric

(d) EUR evaluation metric

Fig. 9. MONTAGE: energy, reliability, CO_2, and EUR evaluation metrics.

and EBABC-PF. It shows that all algorithms experience higher reliability scores as application numbers increase. However, A3C-CGT consistently maintains higher server and link reliability. At smaller workload levels, A3C-CGT achieves reliability scores 56, whereas MSISCSOA scores 51, SDAR-NFAT scores 47, DCOHHOTS scores 45, A2C scores 42, DQN scores 40, and EBABC-PF scores 37. At extreme workload levels, these scores increase to 91 for A3C-CGT, 82 for MSISCSOA, 79 for SDAR-NFAT, 75 for DCOHHOTS, 72 for A2C, 70 for DQN, and 68 for EBABC-PF.

Fig. 9(c) focuses on carbon dioxide (CO2) emissions, which are directly linked to energy consumption. Lower energy consumption results in reduced CO2 emissions. The data in Fig. 9(c) demonstrates that A3C-CGT exhibits lower CO2 emissions compared to MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF. For example, at smaller workload levels, A3C-CGT emits 0.10 units of CO2, while MSISCSOA consumes 0.12, SDAR-NFAT consumes 0.15, DCOHHOTS emits 0.17, A2C emits 0.19, DQN emits 0.21, and EBABC-PF emits 0.24. As workload levels increase to extreme levels, these emissions rise to 0.34 for A3C-CGT, 0.37 for MSISCSOA, 0.39 for SDAR-NFAT, 0.42 for DCOHHOTS, 0.45 for A2C, 0.47 for DQN, and 0.49 for EBABC-PF.

Fig. 9(d) compares the A3C-CGT algorithm with MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF using the EUR metric across different application workloads. At smaller application levels, A3C-CGT demonstrates superior EUR effectiveness compared to MSISCSOA by 237, SDAR-NFAT by 247, DCOHHOTS by 251, A2C by 257, DQN by 271, and EBABC-PF by 277. As application levels increase to extreme levels, A3C-CGT maintains substantial improvement margins of 512 over MSISCSOA, 517 over SDAR-NFAT, 519 over DCOHHOTS, 520 over A2C, 590 over DQN, and 672 over EBABC-PF, respectively.

8. Statistical analysis

This section analyzes the A3C-CGT algorithm's effectiveness, comparing it with MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF. The algorithm was executed multiple times to ensure statistical accuracy.

The first part of the section examines the statistical data across all modes concerning energy, reliability, CO2 emissions, and EUR using two significant real-world workflow datasets: Cybershake and Montage.

In the second part, the Wilcoxon rank-sum and Friedman statistical tests are used to analyze and evaluate the experimental results.

8.1. Statistical analysis: Cybershake vs. Montage

For this evaluation, we defined five workload layers, labeled from $\mathcal{L}_1$ to $\mathcal{L}_5$, with each layer containing three distinct workload sizes. LT represents low computational complexity, ML denotes moderate

Table 4
CYBERSHAKE: compared algorithms with respect to energy, reliability, $C\mathcal{O}_2$, and EUR.

(a) Comparing algorithms with respect to energy consumption using different utilization loads

Algorithms	$\mathcal{L}_1$			$\mathcal{L}_2$			$\mathcal{L}_3$			$\mathcal{L}_4$			$\mathcal{L}_5$		
	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH
A3C-CGT	7.21	7.64	8.45	8.89	8.96	9.12	8.12	8.45	8.92	7.64	7.92	8.12	6.14	6.94	7.54
MSISCSOA	7.37	7.72	8.51	8.92	8.98	9.19	8.37	8.57	8.97	7.77	7.98	8.32	6.21	6.96	7.81
SDAR-NFAT	7.52	7.87	8.63	8.96	8.99	9.23	8.43	8.64	8.98	7.88	7.99	8.42	6.37	6.97	7.94
DCOHHOTS	7.63	7.92	8.75	8.98	8.99	9.41	8.55	8.73	8.99	7.91	7.99	8.55	6.56	6.99	7.98
A2C	7.63	8.12	8.86	8.99	9.12	9.64	8.87	8.97	9.14	7.92	8.10	8.47	6.78	7.15	8.21
DQN	8.64	8.91	9.14	9.15	10.54	10.82	9.12	9.66	9.89	8.14	8.72	8.92	7.19	7.92	8.77
EBABC-PF	9.12	9.66	9.89	9.92	10.82	10.92	9.87	10.1	10.2	8.72	9.12	9.66	7.82	8.45	9.67

(b) Comparing algorithms with respect to reliability using different utilization loads

Algorithms	$\mathcal{L}_1$			$\mathcal{L}_2$			$\mathcal{L}_3$			$\mathcal{L}_4$			$\mathcal{L}_5$		
	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH
A3C-CGT	0.20	0.36	0.65	0.35	0.39	0.42	0.68	0.75	0.81	0.74	0.79	0.84	0.84	0.89	0.94
MSISCSOA	0.18	0.34	0.62	0.32	0.36	0.40	0.66	0.72	0.80	0.72	0.77	0.82	0.82	0.87	0.92
SDAR-NFAT	0.17	0.32	0.61	0.30	0.34	0.38	0.64	0.70	0.78	0.71	0.75	0.81	0.81	0.85	0.91
DCOHHOTS	0.16	0.30	0.60	0.29	0.33	0.37	0.63	0.69	0.76	0.70	0.75	0.81	0.81	0.85	0.90
A2C	0.16	0.26	0.35	0.28	0.32	0.36	0.62	0.68	0.75	0.69	0.75	0.81	0.81	0.82	0.87
DQN	0.15	0.21	0.31	0.27	0.29	0.35	0.62	0.67	0.75	0.69	0.72	0.77	0.75	0.80	0.81
EBABC-PF	0.12	0.18	0.21	0.21	0.25	0.31	0.58	0.65	0.72	0.67	0.68	0.72	0.71	0.78	0.79

(c) Comparing algorithms with respect to $C\mathcal{O}_2$ using different utilization loads

Algorithms	$\mathcal{L}_1$			$\mathcal{L}_2$			$\mathcal{L}_3$			$\mathcal{L}_4$			$\mathcal{L}_5$		
	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH
A3C-CGT	5.21	5.47	6.45	9.64	9.45	10.2	10.2	10.6	10.8	8.45	9.64	9.95	7.45	7.89	8.64
MSISCSOA	6.32	6.42	7.12	10.3	10.5	11.7	11.3	11.7	11.9	9.12	10.3	10.5	8.6	8.8	9.6
SDAR-NFAT	6.54	6.67	7.32	10.6	10.7	11.8	11.9	12.3	12.9	10.2	10.6	10.7	9.2	9.8	9.9
DCOHHOTS	7.21	7.31	7.45	11.2	11.5	12.2	12.3	12.8	13.3	11.6	11.2	11.4	9.6	10.2	10.4
A2C	8.32	8.47	8.75	11.6	11.8	13.2	13.4	13.8	13.9	12.2	12.5	12.8	10.2	11.5	11.8
DQN	8.67	8.87	9.25	12.2	12.7	13.8	13.9	14.3	14.5	13.6	13.8	13.9	11.5	11.8	12.4
EBABC-PF	9.11	9.77	10.5	13.2	13.8	14.2	14.2	14.8	15.2	14.2	14.7	14.9	12.7	12.9	13.5

(d) Comparing algorithms with respect to EUR using different utilization loads

Algorithms	$\mathcal{L}_1$			$\mathcal{L}_2$			$\mathcal{L}_3$			$\mathcal{L}_4$			$\mathcal{L}_5$		
	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH
A3C-CGT	0.70	0.72	0.76	0.72	0.75	0.81	0.74	0.79	0.84	0.80	0.82	0.87	0.82	0.87	0.92
MSISCSOA	0.68	0.70	0.74	0.70	0.73	0.80	0.72	0.77	0.82	0.78	0.81	0.85	0.80	0.85	0.90
SDAR-NFAT	0.66	0.69	0.72	0.68	0.71	0.78	0.70	0.75	0.80	0.76	0.80	0.83	0.78	0.83	0.88
DCOHHOTS	0.65	0.67	0.71	0.67	0.70	0.76	0.69	0.73	0.78	0.75	0.79	0.81	0.76	0.81	0.86
A2C	0.64	0.66	0.70	0.66	0.69	0.75	0.68	0.72	0.77	0.74	0.78	0.80	0.75	0.80	0.85
DQN	0.61	0.64	0.67	0.64	0.67	0.73	0.66	0.70	0.76	0.72	0.76	0.78	0.73	0.78	0.83
EBABC-PF	0.60	0.62	0.65	0.62	0.65	0.71	0.64	0.68	0.74	0.70	0.74	0.76	0.71	0.76	0.81

computational complexity, and LH indicates the highest computational complexity. A3C-CGT is compared to MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF in Table 4a through c. They show how much energy they use, how reliable they are, how much CO2 they release, and how much EUR they cost.

Table 4a highlights A3C-CGT's significant energy savings across all problem sizes, partially due to the application of DVFS techniques. Table 4b focuses on reliability, showing A3C-CGT consistently outperforming the other algorithms, even under higher workload conditions. Table 4c shows CO2 emissions, showing that A3C-CGT is better than the other algorithms at lowering emissions through innovative server energy management and efficient task assignment. Table 4d evaluates the EUR metric, revealing A3C-CGT's higher efficiency significantly as task frequencies increase.

Similarly, Table 5a to d compare A3C-CGT with the same set of algorithms using the Montage workflow dataset. The results mirror those from the Cybershake dataset: Table 4a confirms A3C-CGT's significant energy savings; Table 4b shows its superior reliability; Table 4c demonstrates its ability to minimize CO2 emissions; and Table 4d highlights its higher efficiency in the EUR metric.

Overall, these results emphasize A3C-CGT's robustness and efficiency in cloud resource management and performance optimization compared to MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF.

8.2. Statistical analysis: Friedman vs. Wilcoxon

Table 6a presents the results of Wilcoxon's signed-rank test, a nonparametric test conducted with a confidence level of 0.05 for the ρ value. This test was used to validate the effectiveness of the A3C-CGT paradigm and to show that the results have a statistically significant difference when compared to MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF.

Additionally, to further demonstrate the performance of all algorithms, we performed the Friedman evaluation test, as shown in Table 6b, to assess the overall optimization achieved by each algorithm. According to the results summarized in , the A3C-CGT paradigm significantly outperforms MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF in all benchmark evaluations based on the Friedman and Wilcoxon tests.

9. A3C-CGT validation

We conducted additional tests on the algorithms to confirm the effectiveness of the A3C-CGT paradigm. In this scenario, comprising two main stages, we initially analyzed a coalitional group ($C\mathcal{G}$) of 350 computing servers and 750 virtual machines. We used six evaluation metrics: energy, reliability, CO2 emissions, bandwidth, productivity, EUR, QoS, and resources deficiency. Each server was trained under different utilization loads.

Table 5
MONTAGE: compared algorithms with respect to energy, reliability, CO_2, and EUR.

(a) Comparing algorithms with respect to energy consumption using different utilization loads

Algorithms	$\mathcal{L}_1$			$\mathcal{L}_2$			$\mathcal{L}_3$			$\mathcal{L}_4$			$\mathcal{L}_5$		
	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH
A3C-CGT	16.5	16.8	17.4	15.6	15.8	16.7	14.5	14.7	15.7	13.6	13.9	13.8	12.4	12.7	13.2
MSISCSOA	16.8	16.9	17.8	16.2	16.7	17.2	15.8	15.9	16.2	14.2	14.6	14.8	13.2	13.5	13.8
SDAR-NFAT	16.9	17.2	18.2	16.8	17.2	17.8	16.3	16.8	17.4	14.7	14.9	15.2	13.7	14.1	14.6
DCOHHOTS	17.2	17.7	18.5	17.2	17.8	18.2	16.8	17.2	17.8	15.2	15.8	15.9	14.3	14.7	14.9
A2C	17.5	18.2	18.8	17.5	17.9	18.6	17.2	17.5	18.2	15.6	16.2	16.5	14.7	14.9	15.2
DQN	17.9	18.7	19.2	17.8	18.2	18.9	17.6	17.8	18.8	15.9	16.7	16.9	15.3	15.5	15.8
EBABC-PF	18.2	19.4	19.7	18.4	18.7	19.3	17.9	18.3	19.4	16.5	16.9	17.3	16.5	16.8	17.9

(b) Comparing algorithms with respect to reliability using different utilization loads

Algorithms	$\mathcal{L}_1$			$\mathcal{L}_2$			$\mathcal{L}_3$			$\mathcal{L}_4$			$\mathcal{L}_5$		
	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH
A3C-CGT	0.61	0.65	0.68	0.71	0.72	0.74	0.76	0.80	0.81	0.81	0.83	0.85	0.89	0.95	0.97
MSISCSOA	0.59	0.63	0.66	0.69	0.70	0.72	0.74	0.78	0.79	0.79	0.81	0.83	0.87	0.93	0.95
SDAR-NFAT	0.57	0.61	0.64	0.67	0.68	0.70	0.72	0.76	0.77	0.77	0.79	0.81	0.85	0.91	0.93
DCOHHOTS	0.55	0.59	0.62	0.65	0.66	0.68	0.70	0.74	0.75	0.73	0.77	0.79	0.83	0.89	0.91
A2C	0.53	0.57	0.60	0.63	0.64	0.66	0.68	0.72	0.73	0.71	0.75	0.77	0.81	0.87	0.89
DQN	0.51	0.55	0.58	0.61	0.62	0.64	0.66	0.70	0.71	0.69	0.73	0.75	0.79	0.85	0.87
EBABC-PF	0.49	0.53	0.56	0.59	0.60	0.62	0.64	0.68	0.69	0.67	0.71	0.73	0.77	0.83	0.85

(c) Comparing algorithms with respect to CO_2 using different utilization loads

Algorithms	$\mathcal{L}_1$			$\mathcal{L}_2$			$\mathcal{L}_3$			$\mathcal{L}_4$			$\mathcal{L}_5$		
	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH
A3C-CGT	3.25	4.78	7.85	10.2	10.8	11.2	11.4	11.7	12.4	11.9	12.4	12.8	12.6	12.8	13.6
MSISCSOA	3.45	4.93	7.92	11.4	11.7	12.7	11.7	12.5	12.9	12.4	12.8	13.4	13.7	13.9	14.3
SDAR-NFAT	4.12	5.32	8.32	11.9	12.5	13.2	12.5	12.9	13.4	12.9	13.4	13.8	14.3	14.8	15.7
DCOHHOTS	5.32	5.73	8.87	12.3	12.9	13.7	13.4	13.9	14.5	13.3	13.8	13.9	14.6	14.9	16.3
A2C	5.78	6.31	9.23	13.5	13.9	14.3	13.8	14.4	15.3	13.9	14.2	14.5	15.3	15.6	16.8
DQN	6.24	6.77	10.13	14.2	14.7	15.5	14.2	14.8	16.5	14.3	14.8	15.1	15.9	16.3	17.2
EBABC-PF	6.64	7.32	10.65	14.9	15.3	16.2	14.8	15.3	17.6	14.8	15.4	15.8	16.3	16.8	17.9

(d) Comparing algorithms with respect to EUR using different utilization loads

Algorithms	$\mathcal{L}_1$			$\mathcal{L}_2$			$\mathcal{L}_3$			$\mathcal{L}_4$			$\mathcal{L}_5$		
	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH	LT	ML	LH
A3C-CGT	0.85	0.88	0.92	0.87	0.88	0.94	0.90	0.92	0.94	0.92	0.95	0.97	0.94	0.96	0.97
MSISCSOA	0.83	0.86	0.90	0.85	0.86	0.92	0.88	0.90	0.92	0.90	0.93	0.95	0.92	0.94	0.95
SDAR-NFAT	0.80	0.83	0.87	0.82	0.83	0.89	0.79	0.89	0.89	0.87	0.90	0.92	0.89	0.91	0.92
DCOHHOTS	0.78	0.81	0.85	0.80	0.81	0.87	0.77	0.87	0.88	0.85	0.88	0.90	0.87	0.89	0.90
A2C	0.76	0.79	0.83	0.78	0.79	0.85	0.75	0.85	0.86	0.83	0.86	0.88	0.85	0.87	0.88
DQN	0.74	0.77	0.81	0.76	0.77	0.83	0.73	0.83	0.84	0.81	0.84	0.86	0.83	0.85	0.86
EBABC-PF	0.72	0.77	0.79	0.74	0.75	0.81	0.71	0.81	0.82	0.79	0.82	0.84	0.81	0.83	0.84

In the second stage, we utilized a workflow module generator ($\mathcal{WG}$) to produce various application datasets with different data sizes and computational complexities. We then randomly selected six problem indexes, each with specific deadlines and dynamic preemptive characteristics. We used the evaluation metrics of energy, reliability, CO2 emissions, bandwidth, QoS, and productivity to assess the performance of the algorithms A3C-CGT, MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF.

9.1. Validation under CG consideration

For this evaluation, we chose three notable algorithms – MSISCSOA, SDAR-NFAT, and DQN – to compare with the A3C-CGT model. In Fig. 10(a), the A3C-CGT algorithm is evaluated against MSISCSOA, SDAR-NFAT, and DQN algorithms in terms of energy consumption. The A3C-CGT algorithm consistently outperforms its competitors across several benchmark problems. For instance, when executed on a set of computing resources, the A3C-CGT algorithm achieved an energy consumption value of 3.31, whereas the MSISCSOA model recorded a rate of 4.25, SDAR-NFAT reached 4.35, and DQN obtained 5.25, respectively.

Fig. 10(b) compares the A3C-CGT algorithm with MSISCSOA, SDAR-NFAT, and DQN algorithms in terms of reliability. The numerical data analysis shows that the A3C-CGT algorithm consistently outperforms its competitors across benchmark problems. For example, when executed on a set of computing resources, the A3C-CGT algorithm achieved a high-reliability value of 80.21, while the MSISCSOA model recorded a rate of 67.51, SDAR-NFAT reached 57.28, and DQN obtained 47.64. Additionally, in Fig. 10(c), we compared the A3C-CGT algorithm with MSISCSOA, SDAR-NFAT, and DQN algorithms in terms of CO2 emissions. The numerical data analysis reveals that the A3C-CGT algorithm consistently outperforms its competitors across various benchmark problems. For instance, when executed on a set of computing resources, the A3C-CGT algorithm achieved a CO2 emission value of 1.15, while the MSISCSOA model recorded 2.43, SDAR-NFAT reached 2.85, and DQN obtained 3.37.

As shown in Fig. 10(d), we conducted a direct comparison of the bandwidth dependency of the A3C-CGT algorithm with that of the MSISCSOA, SDAR-NFAT, and DQN algorithms. The numerical data analysis consistently demonstrates the A3C-CGT algorithm's superiority over its competitors across various benchmark problems. For instance, when executed on a set of computing resources, the A3C-CGT algorithm achieved a bandwidth dependency value of 0.85, while the MSISCSOA model recorded 0.63, SDAR-NFAT reached 0.52, and DQN obtained 0.48.

Fig. 10(e) displays the service productivity percentages for the A3C-CGT, MSISCSOA, SDAR-NFAT, and DQN algorithms. The A3C-CGT algorithm outperforms MSISCSOA, SDAR-NFAT, DCOHHOTS, and DQN across benchmark problems. The results indicate that the A3C-CGT algorithm had the highest service productivity efficiency at 81.31,

Table 6
Friedman evaluation and Wilcoxon signed rank evaluation.

(a) Friedman data analysis.

Problem	Mean rank						
Index	A3C-CGT	MSISCSOA	SDAR-NFAT	DCOHHOTS	A2C	DQN	EBABC-PF
01	2.43	2.77	2.82	2.88	2.94	3.15	3.27
02	2.13	2.43	2.51	2.75	2.87	2.93	3.11
03	2.72	2.78	2.82	2.93	3.66	3.72	3.81
04	2.52	2.61	2.66	2.73	2.84	2.93	3.14
05	2.61	2.69	2.74	2.81	2.96	3.14	3.44
06	3.08	3.12	3.44	3.64	3.78	3.81	3.84
07	2.27	2.34	2.47	3.52	3.66	3.72	4.12
08	2.14	2.25	2.37	3.21	3.34	3.71	3.83
09	3.11	3.15	3.57	3.71	3.78	3.83	3.91
10	2.41	2.62	2.68	2.77	3.21	3.45	3.67

(b) Wilcoxon Signed Rank data analysis

Problem	A3C-CGT vs. MSISCSOA		A3C-CGT vs. SDAR-NFAT		A3C-CGT vs. DCOHHOTS		A3C-CGT vs. A2C		A3C-CGT vs. DQN	
Index	Sign rank	ρ	Sign rank	ρ	Sign rank	ρ	Sign rank	ρ	Sign rank	ρ
01	591	0.023	642	0.012	934	0.009	408	0.023	1133	0.034
02	485	0.020	621	0.012	1021	0.009	1187	0.018	931	0.005
03	1037	0.032	945	0.042	911	0.032	1021	0.013	942	0.043
04	1031	0.013	834	0.008	943	0.032	1031	0.045	1905	0.032
05	1321	0.025	944	0.022	1321	0.034	1354	0.017	954	0.025
06	1421	0.032	1401	0.031	1283	0.013	1077	0.023	1456	0.014
07	965	0.043	1055	0.013	1044	0.032	965	0.024	1075	0.027
08	1321	0.033	967	0.027	1034	0.027	1319	0.035	944	0.034
09	932	0.013	1032	0.013	1376	0.021	1704	0.036	1032	0.032
10	1043	0.027	1327	0.024	1031	0.029	1054	0.026	971	0.035

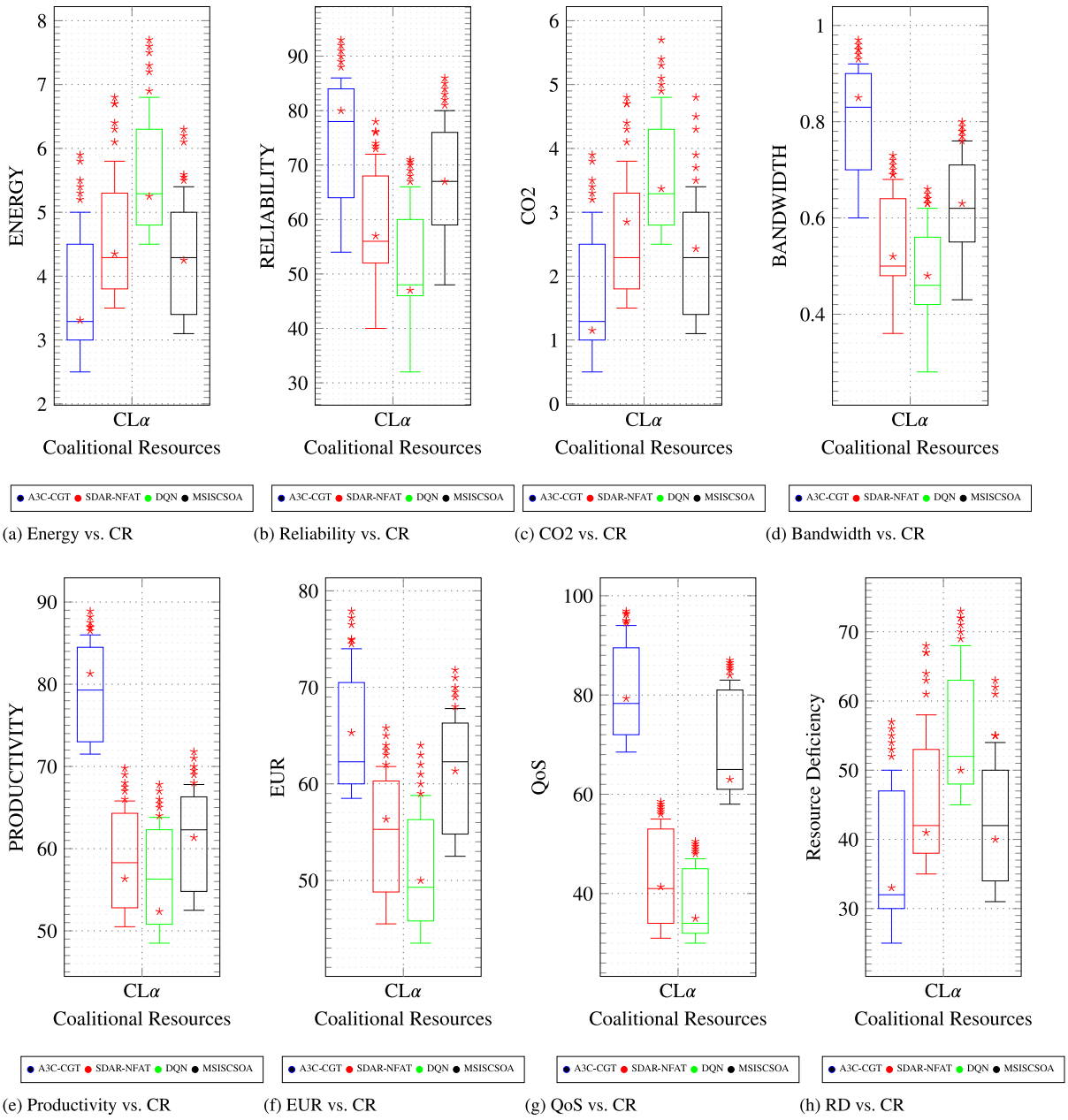
followed by the MSISCSOA model with 61.35, SDAR-NFAT with 56.35, and DQN with 52.07. These algorithms were run on a set of computing resources. As depicted in Fig. 10(f), we compared the EUR values of the A3C-CGT algorithm with those of the MSISCSOA, SDAR-NFAT, DCOHHOTS, and DQN algorithms. The numerical data analysis consistently highlights the A3C-CGT algorithm's superiority over its competitors across various benchmark problems. For example, when executed on a set of computing resources, the A3C-CGT algorithm achieved a EUR value of 65.31, while the MSISCSOA model recorded 61.01, SDAR-NFAT reached 56.14, and DQN obtained 50.12.

In Fig. 10(g), the A3C-CGT algorithm is evaluated against MSISCSOA, SDAR-NFAT, and DQN algorithms in terms of QoS. The A3C-CGT algorithm consistently outperforms its competitors across several benchmark problems. For instance, when executed on a set of computing resources, the A3C-CGT algorithm achieved a quality of service value of 79.49. In contrast, the MSISCSOA model documented a rate of 63.41, SDAR-NFAT reached 41.08, and DQN obtained 35.74, respectively. In Fig. 10(h), the A3C-CGT algorithm is evaluated against the MSISCSOA, SDAR-NFAT, and DQN algorithms in terms of resource deficiency. The A3C-CGT algorithm consistently outperforms its competitors across various benchmark problems. For example, when executed on a set of computing resources, the A3C-CGT algorithm achieved a resource deficiency value of 33, whereas the MSISCSOA model recorded 40, SDAR-NFAT reached 41, and DQN obtained 50.

9.2. Validation under $\mathcal{WG}$ consideration

In Fig. 11(a), the A3C-CGT algorithm is compared to the MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF algorithms in terms of energy consumption. The A3C-CGT algorithm consistently surpasses its competitors. For instance, with 30 modules, the A3C-CGT model consumes about 3.09 units of energy, while MSISCSOA uses 4.05, SDAR-NFAT uses 4.54, DCOHHOTS uses 5.04, A2C uses 5.24, DQN uses 6.06, and EBABC-PF uses 6.57. When the number of modules increases to 50, the A3C-CGT model's consumption rises to approximately 5.01, compared to 6.08 for MSISCSOA, 6.71 for SDAR-NFAT, 7.05 for DCOHHOTS, 7.54 for A2C, 8.21 for DQN, and 9.06 for EBABC-PF. At 80 modules, the A3C-CGT model's consumption reaches around 21.05, while MSISCSOA consumes 24.04, SDAR-NFAT consumes 25.01, DCOHHOTS consumes 26.05, A2C consumes 28.03, DQN consumes 32.08, and EBABC-PF consumes 37.10. Fig. 11(b) illustrates a comparison of the A3C-CGT algorithm with the MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF algorithms in terms of reliability. The A3C-CGT algorithm consistently outperforms its competitors. For example, with 30 modules, the A3C-CGT model scores about 0.67 units of reliability, whereas MSISCSOA scores 0.67, SDAR-NFAT scores 0.65, DCOHHOTS scores 0.65, A2C scores 0.63, DQN scores 0.62, and EBABC-PF scores 0.62. When the number of modules increases to 50, the A3C-CGT model's reliability slightly decreases to around 0.69, compared to 0.68 for MSISCSOA, 0.67 for SDAR-NFAT, 0.66 for DCOHHOTS, 0.64 for A2C, 0.63 for DQN, and 0.61 for EBABC-PF. At 80 modules, the A3C-CGT model's reliability rises to approximately 0.94, while MSISCSOA scores 0.91, SDAR-NFAT scores 0.89, DCOHHOTS scores 0.88, A2C scores 0.84, DQN scores 0.83, and EBABC-PF scores 0.81.

Fig. 11(c) compares the A3C-CGT algorithm with the MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF algorithms in terms of CO2 emissions. Across all problem indices, the A3C-CGT algorithm consistently achieves lower values than its counterparts. For instance, with 30 modules, the A3C-CGT model has a CO2 emission of about 1.02 units, while MSISCSOA emits 1.17, SDAR-NFAT emits 2.06, DCOHHOTS emits 2.51, A2C emits 3.04, DQN emits 3.07, and EBABC-PF emits 3.09. With 50 modules, the A3C-CGT model's emissions increase slightly to around 2.51, whereas MSISCSOA emits 3.08, SDAR-NFAT emits 5.07, DCOHHOTS emits 7.08, A2C emits 9.10, DQN emits 11.06, and EBABC-PF emits 14.05. At 80 modules, the A3C-CGT model's emissions rise to approximately 18.02 units, while MSISCSOA emits 21.02, SDAR-NFAT emits 23.04, DCOHHOTS emits 25.07, A2C emits 29.06, DQN emits 31.02, and EBABC-PF emits 35.04. In Fig. 11(d), the A3C-CGT algorithm is compared to the MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF algorithms in terms of bandwidth dependency. The A3C-CGT algorithm consistently shows higher values than its competitors. For example, with 30 modules, the A3C-CGT model has a bandwidth dependency of about 47.8 units, while MSISCSOA has 47.05, SDAR-NFAT has 47.01, DCOHHOTS has 45.56, A2C has 46.28, DQN has 45.52, and EBABC-PF has 45.04. With 50 modules, the A3C-CGT model's dependency slightly increases to around 52 units, whereas MSISCSOA has 50, SDAR-NFAT has 49,



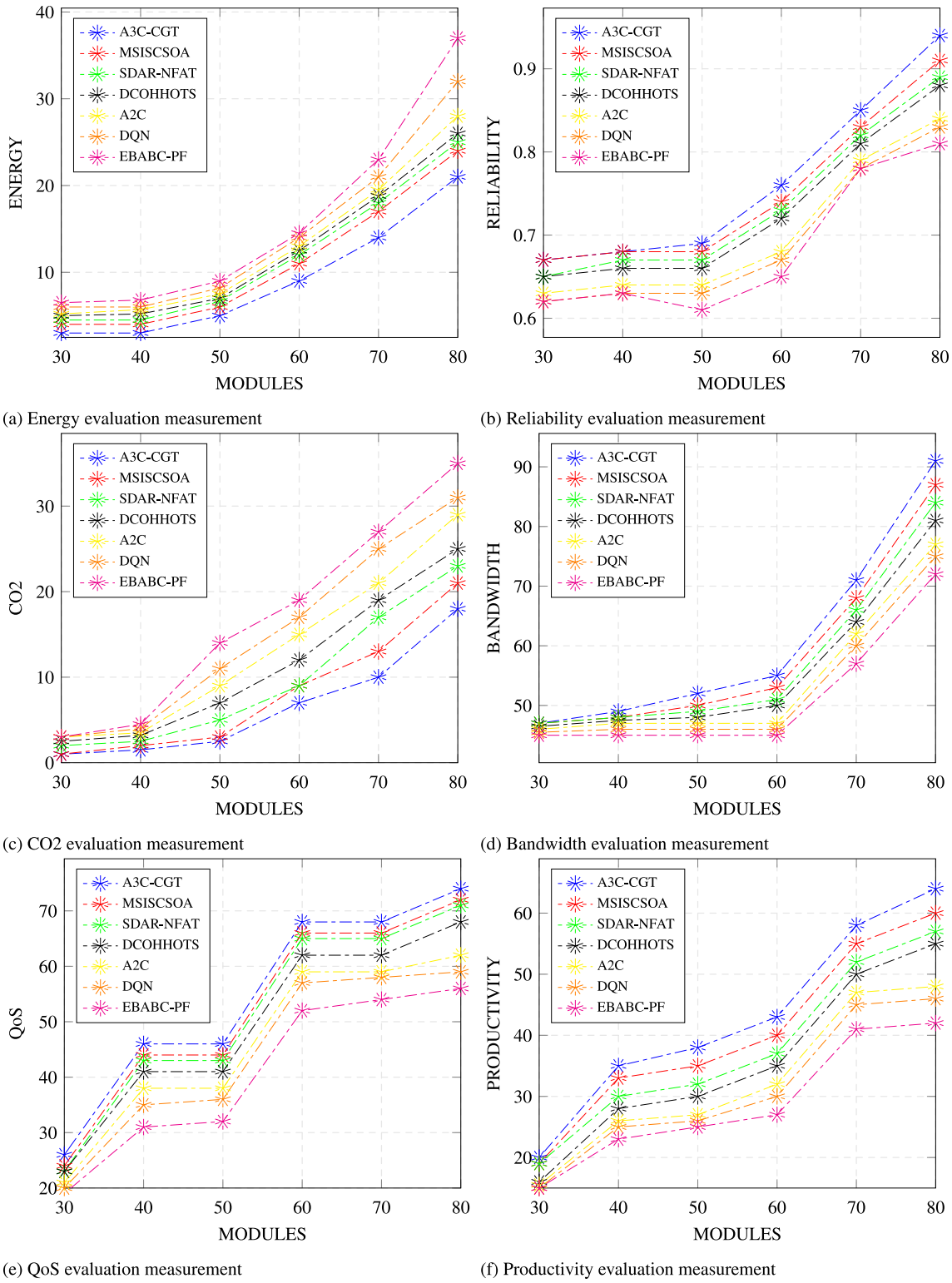
(a) Energy vs. CR (b) Reliability vs. CR (c) CO2 vs. CR (d) Bandwidth vs. CR

(e) Productivity vs. CR (f) EUR vs. CR (g) QoS vs. CR (h) RD vs. CR

Fig. 10. The box plot illustrates the distribution of values, emphasizing the median, quartiles, and outliers. It provides a visual overview of the energy, reliability, CO2 emissions, bandwidth, productivity, EUR, QoS, and resource deficiency data analysis and depicts the spread of execution cost values.

DCOHHOTS has 48, A2C has 47, DQN has 46, and EBABC-PF has 45. At 80 modules, the A3C-CGT model's dependency rises to approximately 91 units, while MSISCSOA has 87, SDAR-NFAT has 84, DCOHHOTS has 81, A2C has 77, DQN has 75, and EBABC-PF has 72.

In Fig. 11(e), the A3C-CGT algorithm is compared to the MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF algorithms in terms of QoS. The A3C-CGT algorithm consistently demonstrates higher values than its competitors. For instance, with 30 modules, the A3C-CGT model achieves a high QoS value of 26, while MSISCSOA has 24, SDAR-NFAT has 23, DCOHHOTS has 23, A2C has 21, DQN has 20, and EBABC-PF has 19. As the number of modules increases to 50, the A3C-CGT model's QoS value rises to approximately 46, compared to 44 for MSISCSOA, 43 for SDAR-NFAT, 41 for DCOHHOTS, 38 for A2C, 36 for DQN, and 32 for EBABC-PF. At 80 modules, the A3C-CGT model's QoS further increases to around 74, while MSISCSOA has 72, SDAR-NFAT has 71, DCOHHOTS has 68, A2C has 62, DQN has 59, and EBABC-PF has 56. In Fig. 11(f), the A3C-CGT algorithm is compared with the MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF algorithms in terms of service productivity. The A3C-CGT algorithm consistently outperforms its competitors. With 30 modules, the A3C-CGT model achieves a high service productivity value of 20, while MSISCSOA records 19, SDAR-NFAT records 19, DCOHHOTS records 16, A2C records 15.5, DQN records 15, and EBABC-PF records 15. As the number of modules increases to 50, the A3C-CGT model's service productivity rises to approximately 38, compared to 35 for MSISCSOA, 32 for SDAR-NFAT, 30 for DCOHHOTS, 27 for A2C, 26 for DQN, and 25 for EBABC-PF. At 80 modules, the A3C-CGT model's service productivity further increases to around 64, while MSISCSOA reaches 60, SDAR-NFAT reaches 57, DCOHHOTS reaches 55, A2C reaches 48, DQN reaches 46, and EBABC-PF reaches 42.



(a) Energy evaluation measurement

(b) Reliability evaluation measurement

(c) CO2 evaluation measurement

(d) Bandwidth evaluation measurement

(e) QoS evaluation measurement

(f) Productivity evaluation measurement

Fig. 11. Impact of WG evaluation metric on algorithm's performance.

10. Discussion

This section thoroughly discusses the performance variations observed with the A3C-CGT paradigm.

Figs. 5(a) and 5(b) assess the performance of the A3C-CGT algorithm under different numbers of applications and VMs, focusing on energy consumption metrics. The A3C-CGT paradigm outperforms other algorithms by employing multiple VM management techniques, efficiently reusing allocated VMs for new applications, and minimizing startup, idle, and shutdown overheads. Additionally, it optimizes VM allocation using a window matrix to avoid overlaps, further enhancing efficiency. In Figs. 5(c) and 5(d), reliability metrics are analyzed across various workloads and VM configurations. A3C-CGT surpasses other algorithms by incorporating the Poisson distribution to account for server and communication link failures, which is critical for reliable task scheduling on cloud resources.

Fig. 6(a) evaluates CO2 emissions, a key focus of this study, demonstrating that A3C-CGT effectively reduces CO2 emissions by scheduling batches of applications on cloud resources. Similarly, Fig. 6(b) evaluates A3C-CGT's performance across different numbers of VMs, highlighting its efficiency in collaborative system operation and energy-conscious task execution. Figs. 6(c) and 6(d) replicate the evaluation using bandwidth dependency metrics. A3C-CGT excels by considering bandwidth failure rates during task allocation to cloud destinations, ensuring optimal bandwidth utilization.

Figs. 7(a) through 7(d) evaluate key metrics such as service productivity, EUR, and QoS across different applications and VM scenarios. A3C-CGT consistently outperforms MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF models by optimizing the performance-to-power ratio for each server and minimizing overhead through efficient VM management and preemptive time slot allocation. The effectiveness of A3C-CGT is further demonstrated through evaluations of real scientific workflow applications like Cybershake and Montage, where it consistently shows superior resource utilization across energy, reliability, CO2 emissions, and EUR metrics.

Statistical analysis in Table 4a highlights A3C-CGT's energy efficiency compared to MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF, with significant energy savings across all problem sizes, partly due to the application of DVFS techniques. Table 4b focuses on reliability metrics, showing that A3C-CGT consistently outperforms its competitors by maintaining higher reliability even with increased workloads. Table 4c addresses CO2 emissions, demonstrating A3C-CGT's capability to minimize emissions through strategic server energy management and efficient task assignment. Table 4d evaluates the EUR metric, where A3C-CGT achieves greater efficiency than MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF, significantly as task frequencies increase.

Table 6a displays the results of Wilcoxon's signed-rank test, indicating a statistically significant difference between the performance of A3C-CGT and the algorithms MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF. Table 6b employs the Friedman test to evaluate the overall optimization effectiveness of each algorithm. The findings summarized in demonstrate that the A3C-CGT paradigm significantly surpasses MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF across all benchmark evaluations, as confirmed by both the Friedman and Wilcoxon tests.

Table 5a through d further support A3C-CGT's superiority in energy conservation, reliability enhancement, CO2 emission reduction, and EUR improvement across various application scenarios and workload sizes.

To verify the effectiveness of the A3C-CGT paradigm, further tests were carried out in two stages: validation under CG conditions and validation under WG conditions, as depicted in Figs. 10(a) through 11(f). The analysis of these tests reveals that the proposed A3C-CGT approach outperforms MSISCSOA, SDAR-NFAT, DCOHHOTS, A2C, DQN, and EBABC-PF in terms of energy consumption, reliability, CO2 emissions, bandwidth, productivity, EUR, QoS, and resource deficiency.

Overall, these results emphasize A3C-CGT's robustness and efficiency in managing cloud resources and optimizing performance metrics compared to existing scheduling algorithms.

11. Conclusions and future work

As the number of data centers continues to rise, cloud computing, which has become increasingly popular for data management and storage among individuals and organizations globally, faces significant challenges. Enhancing the efficiency of workflow scheduling has thus become crucial. When selecting the optimal application servers, emphasizing performance and power efficiency is essential. This process demands careful consideration of factors such as energy consumption, carbon emissions, scheduling reliability, and QoS.

Many existing studies struggle to optimize workflow scheduling, especially when trying to balance energy and CO2 efficiency with system reliability. These challenges are often due to the complexity of resource allocation, the dynamic nature of workloads, and the need for real-time decision-making to avoid system failures while minimizing energy usage. However, relying on a single optimization algorithm is insufficient to address these competing objectives; instead, hybridizing multiple optimization algorithms is necessary to achieve optimal solutions without increasing computational complexity or falling into local optima.

In this paper, an innovative model called A3C-CGT is introduced, which hybridizes an enhanced asynchronous advantage actor–critic (A3C) with a new coalitional game theory technique known as merge-and-split-based to optimize these conflicting objectives simultaneously while adhering to SLAs. This model operates through three key stages. The first stage focuses on forming robust server clusters optimized for performance by executing predefined tasks while reducing energy consumption and determining the most efficient way to operate these clusters, including identifying the controller server with the highest efficiency-to-utilization ratio. The second stage identifies the best VM destinations within the search domain, ensuring they provide sufficient processing power and proper execution while maximizing energy efficiency, reducing CO2 emissions, and maintaining reliability, QoS, and SLA compliance. The third stage involves maintaining computing resources in the data center through two primary operations: defining a dynamic, adaptable threshold to identify overloaded and underloaded computing nodes and selecting and allocating VMs to optimize performance and resource utilization across available computing nodes.

Experiments conducted on both real and synthesized workflows demonstrate that the proposed algorithm can learn high-quality scheduling policies for various workflows and optimization objectives, leading to energy efficiency improvements of 7.65% to 19.32%, carbon emission reductions of 3.13% to 14.76%, and reliability enhancements of 17.22% to 41.65%.

Future research will explore specialized hardware and software techniques, particularly in data-driven fields like IoT applications in healthcare and blockchain-driven optimization. Advancing healthcare infrastructure through virtual reality (VR) and artificial intelligence (AI) aims to tackle critical health challenges such as heart attacks and COVID-19. Additionally, integrating blockchain with cloud computing holds promise, ensuring secure data exchange and addressing issues like data privacy and anonymity through innovative methods such as anchoring cloud assets with PoW hash values.

CRediT authorship contribution statement

Mustafa Ibrahim Khaleel: Writing – review & editing, Writing – original draft, Validation, Software, Resources, Methodology, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] E.H. Houssein, A.G. Gad, Y.M. Wazery, P.N. Suganthan, Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends, Swarm Evol. Comput. 62 (2021) 100841, http://dx.doi.org/10.1016/J.SWEVO.2021.100841.

[2] A.S. Abohamama, E. Hamouda, A hybrid energy–Aware virtual machine placement algorithm for cloud environments, Expert Syst. Appl. 150 (2020) 113306, http://dx.doi.org/10.1016/J.ESWA.2020.113306.

[3] L. Barbulescu, J.P. Watson, L.D. Whitley, A.E. Howe, Scheduling space-ground communications for the air force satellite control network, J. Sched. 7 (2004) 7–34, http://dx.doi.org/10.1023/B:JOSH.0000013053.32600.3C/METRICS, URL https://link.springer.com/article/10.1023/B:JOSH.0000013053.32600.3c.

[4] T. Wang, Q. Luo, L. Zhou, G. Wu, Space division and adaptive selection strategy based differential evolution algorithm for multi-objective satellite range scheduling problem, Swarm Evol. Comput. 83 (2023) 101396, http://dx.doi.org/10.1016/J.SWEVO.2023.101396.

[5] S.S. Gill, H. Wu, P. Patros, C. Ottaviani, P. Arora, V.C. Pujol, D. Haunschild, A.K. Parlikad, O. Cetinkaya, H. Lutfiyya, V. Stankovski, R. Li, Y. Ding, J. Qadir, A. Abraham, S.K. Ghosh, H.H. Song, R. Sakellariou, O. Rana, J.J. Rodrigues, S.S. Kanhere, S. Dustdar, S. Uhlig, K. Ramamohanarao, R. Buyya, Modern computing: Vision and challenges, Telemat. Inform. Rep. 13 (2024) 100116, http://dx.doi.org/10.1016/J.TELER.2024.100116.

[6] M. Dayarathna, Y. Wen, R. Fan, Data center energy consumption modeling: A survey, IEEE Commun. Surv. Tutor. 18 (2016) 732–794, http://dx.doi.org/10.1109/COMST.2015.2481183.

[7] R.B. Mitchell, R. York, Reducing the web's carbon footprint: Does improved electrical efficiency reduce webserver electricity use? Energy Res. Soc. Sci. (2020) http://dx.doi.org/10.1016/j.erss.2020.101474.

[8] J.G. Koomey, Worldwide electricity used in data centers, Environ. Res. Lett. (2008) http://dx.doi.org/10.1088/1748-9326/3/3/034008, URL http://www.koomey.com.

[9] A. Varasteh, M. Goudarzi, Server consolidation techniques in virtualized data centers: A survey, IEEE Syst. J. 11 (2017) 772–783, http://dx.doi.org/10.1109/JSYST.2015.2458273.

[10] B. Magotra, D. Malhotra, A.K. Dogra, Adaptive computational solutions to energy efficiency in cloud computing environment using VM consolidation, Arch. Comput. Methods Eng. 30 (2022) 1789–1818, http://dx.doi.org/10.1007/S11831-022-09852-2, URL https://link.springer.com/article/10.1007/s11831-022-09852-2, 2022 30:3.

[11] M. Avgerinou, P. Bertoldi, L. Castellazzi, Trends in data centre energy consumption under the European code of conduct for data centre energy efficiency, Energies 10 (2017) 1470, http://dx.doi.org/10.3390/EN10101470, URL https://www.mdpi.com/1996-1073/10/10/1470/htm https://www.mdpi.com/1996-1073/10/10/1470, 2017, Vol. 10, Page 1470.

[12] M. Uddin, Y. Darabidarabkhani, A. Shah, J. Memon, Evaluating power efficient algorithms for efficiency and carbon emissions in cloud data centers: A review, Renew. Sustain. Energy Rev. 51 (2015) 1553–1563, http://dx.doi.org/10.1016/J.RSER.2015.07.061.

[13] L.A. Barroso, U. Hölzle, The case for energy-proportional computing, Computer 40 (2007) 33–37, http://dx.doi.org/10.1109/MC.2007.443.

[14] P. Jayalakshmi, S.S. Ramesh, Multi-strategy improved sand cat optimization algorithm-based workflow scheduling mechanism for heterogeneous edge computing environment, Sustain. Comput.: Inform. Syst. 43 (2024) 101014, http://dx.doi.org/10.1016/J.SUSCOM.2024.101014.

[15] A.K. Sangaiah, A. Javadpour, P. Pinto, S. Rezaei, W. Zhang, Enhanced resource allocation in distributed cloud using fuzzy meta-heuristics optimization, Comput. Commun. 209 (2023) 14–25, http://dx.doi.org/10.1016/J.COMCOM.2023.06.018.

[16] R. Ghafari, N. Mansouri, Improved Harris Hawks Optimizer with chaotic maps and opposition-based learning for task scheduling in cloud environment, Cluster Comput. 27 (2024) 1421–1469, http://dx.doi.org/10.1007/S10586-023-04021-X/TABLES/23, URL https://link.springer.com/article/10.1007/s10586-023-04021-x.

[17] Z. Wang, S. Chen, L. Bai, J. Gao, J. Tao, R.R. Bond, M.D. Mulvenna, Reinforcement learning based task scheduling for environmentally sustainable federated cloud computing, J. Cloud Comput. 12 (2023) 1–17, http://dx.doi.org/10.1186/S13677-023-00553-0/FIGURES/10, URL https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-023-00553-0.

[18] H. Li, J. Huang, B. Wang, Y. Fan, Weighted double deep Q-network based reinforcement learning for bi-objective multi-workflow scheduling in the cloud, Cluster Comput. 25 (2022) 751–768, http://dx.doi.org/10.1007/S10586-021-03454-6/METRICS, URL https://link.springer.com/article/10.1007/s10586-021-03454-6.

[19] M. Zeedan, G. Attiya, N. El-Fishawy, Enhanced hybrid multi-objective workflow scheduling approach based artificial bee colony in cloud computing, Computing 105 (2023) 217–247, http://dx.doi.org/10.1007/S00607-022-01116-Y/FIGURES/18, URL https://link.springer.com/article/10.1007/s00607-022-01116-y.

[20] J. Ding, S. Schulz, L. Shen, U. Buscher, Z. Lü, Energy aware scheduling in flexible flow shops with hybrid particle swarm optimization, Comput. Oper. Res. 125 (2021) 105088, http://dx.doi.org/10.1016/J.COR.2020.105088.

[21] H. Bhagavathi, S. Rathinavelayatham, K. Shanmugaiah, K. Kanagaraj, D. Elangovan, Improved beetle swarm optimization algorithm for energy efficient virtual machine consolidation on cloud environment, Concurr. Comput.: Pract. Exper. 34 (2022) e6828, http://dx.doi.org/10.1002/CPE.6828, URL https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.6828 https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6828 https://onlinelibrary.wiley.com/doi/10.1002/cpe.6828.

[22] X. Liu, J. Wu, L. Chen, L. Zhang, Energy-aware virtual machine consolidation based on evolutionary game theory, Concurr. Comput.: Pract. Exper. 34 (2022) e6830, http://dx.doi.org/10.1002/CPE.6830, URL https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.6830 https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6830 https://onlinelibrary.wiley.com/doi/10.1002/cpe.6830.

[23] H. Xing, J. Zhu, R. Qu, P. Dai, S. Luo, M.A. Iqbal, An ACO for energy-efficient and traffic-aware virtual machine placement in cloud computing, Swarm Evol. Comput. 68 (2022) 101012, http://dx.doi.org/10.1016/J.SWEVO.2021.101012.

[24] M.A. Khan, A cost-effective power-aware approach for scheduling cloudlets in cloud computing environments, J. Supercomput. 78 (2022) 471–496, http://dx.doi.org/10.1007/S11227-021-03894-2/METRICS, URL https://link.springer.com/article/10.1007/s11227-021-03894-2.

[25] M. Grami, An energy-aware scheduling of dynamic workflows using big data similarity statistical analysis in cloud computing, J. Supercomput. 78 (2022) 4261–4289, http://dx.doi.org/10.1007/S11227-021-04016-8/METRICS, URL https://link.springer.com/article/10.1007/s11227-021-04016-8.

[26] G.S. Liu, M. Tu, Y.S. Tang, T.X. Ding, Energy-aware optimization for the two-agent scheduling problem with fuzzy processing times, Int. J. Interact. Des. Manuf. 17 (2022) 237–248, http://dx.doi.org/10.1007/S12008-022-00927-9/METRICS, URL https://link.springer.com/article/10.1007/s12008-022-00927-9.

[27] H. Li, G. Xu, D. Wang, M.C. Zhou, Y. Yuan, A. Alabdulwahab, Chaotic-nondominated-sorting owl search algorithm for energy-aware multi-workflow scheduling in hybrid clouds, IEEE Trans. Sustain. Comput. 7 (2022) 595–608, http://dx.doi.org/10.1109/TSUSC.2022.3144357.

[28] M. Hussain, L.F. Wei, A. Rehman, F. Abbas, A. Hussain, M. Ali, Deadline-constrained energy-aware workflow scheduling in geographically distributed cloud data centers, Future Gener. Comput. Syst. 132 (2022) 211–222, http://dx.doi.org/10.1016/J.FUTURE.2022.02.018.

[29] S. Khurana, G. Sharma, M. Kumar, N. Goyal, B. Sharma, Reliability based workflow scheduling on cloud computing with deadline constraint, Wirel. Pers. Commun. 130 (2023) 1417–1434, http://dx.doi.org/10.1007/S11277-023-10337-Z/METRICS, URL https://link.springer.com/article/10.1007/s11277-023-10337-z.

[30] S. Sobhanayak, MOHBA:multi-objective workflow scheduling in cloud computing using hybrid BAT algorithm, Computing 105 (2023) 2119–2142, http://dx.doi.org/10.1007/S00607-023-01175-9/METRICS, URL https://link.springer.com/article/10.1007/s00607-023-01175-9.

[31] M.I. Khaleel, A dynamic weight–assignment load balancing approach for workflow scheduling in edge-cloud computing using ameliorated moth flame and rock hyrax optimization algorithms, Future Gener. Comput. Syst. 155 (2024) 465–485, http://dx.doi.org/10.1016/J.FUTURE.2024.02.025.

[32] M.I. Khaleel, M. Safran, S. Alfarhood, M. Zhu, Energy-latency trade-off analysis for scientific workflow in cloud environments: The role of processor utilization ratio and mean grey wolf optimizer, Eng. Sci. Technol. Int. J. 50 (2024) 101611, http://dx.doi.org/10.1016/J.JESTCH.2023.101611.

[33] S.S. Mangalampalli, G.R. Karri, S.N. Mohanty, S. Ali, M.I. Khan, S. Abdullaev, S.A. Alqahtani, Multi-objective prioritized task scheduler using improved asynchronous advantage actor critic (a3c) algorithm in multi cloud environment, IEEE Access 12 (2024) 11354–11377, http://dx.doi.org/10.1109/ACCESS.2024.3355092.

[34] X. Ruan, H. Chen, Y. Tian, S. Yin, Virtual machine allocation and migration based on performance-to-power ratio in energy-efficient clouds, Future Gener. Comput. Syst. 100 (2019) 380–394.

[35] X. Xiao, W. Zheng, Y. Xia, X. Sun, Q. Peng, Y. Guo, A workload-aware VM consolidation method based on coalitional game for energy-saving in cloud, IEEE Access 7 (2019) 80421–80430.

[36] R. Zolfaghari, A.M. Rahmani, Virtual machine consolidation in cloud computing systems: Challenges and future trends, Wirel. Pers. Commun. 115 (2020) 2289–2326, http://dx.doi.org/10.1007/S11277-020-07682-8/METRICS, URL https://link.springer.com/article/10.1007/s11277-020-07682-8.

[37] C. Gu, H. Huang, X. Jia, Power metering for virtual machine in cloud computing-challenges and opportunities, IEEE Access 2 (2014) 1106–1116, http://dx.doi.org/10.1109/ACCESS.2014.2358992.

[38] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, K. Li, Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster, Inform. Sci. 319 (2015) 113–131, http://dx.doi.org/10.1016/J.INS.2015.02.023.

[39] R. Buyya, A. Beloglazov, J. Abawajy, Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges, in: Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010), 2010, http://dx.doi.org/10.48550/arxiv.1006.0308, URL https://arxiv.org/abs/1006.0308v1.

[40] D. Kusic, J.O. Kephart, J.E. Hanson, N. Kandasamy, G. Jiang, Power and performance management of virtualized computing environments via lookahead control, Cluster Comput. 12 (2009) 1–15, http://dx.doi.org/10.1007/S10586-008-0070-Y/METRICS, URL https://link.springer.com/article/10.1007/s10586-008-0070-y.

[41] A. Verma, P. Ahuja, A. Neogi, PMapper: Power and migration cost aware application placement in virtualized systems, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), in: LNCS, Vol. 5346, Springer, Berlin, Heidelberg, 2008, pp. 243–264, http://dx.doi.org/10.1007/978-3-540-89856-6_13/COVER, URL https://link.springer.com/chapter/10.1007/978-3-540-89856-6_13.

[42] RaghavendraRamya, RanganathanParthasarathy, TalwarVanish, WangZhikui, ZhuXiaoyun, No "power" struggles, ACM SIGARCH Comput. Archit. News 36 (2008) 48–59, http://dx.doi.org/10.1145/1353534.1346289, URL https://dl.acm.org/doi/10.1145/1353534.1346289.

[43] D. Zhu, R. Melhem, D. Mossé, The effects of energy management on reliability in real-time embedded systems, in: IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD, 2004, pp. 35–40, http://dx.doi.org/10.1109/ICCAD.2004.1382539.

[44] R. Garg, M. Mittal, L.H. Son, Reliability and energy efficient workflow scheduling in cloud environment, Cluster Comput. 22 (2019) 1283–1297, http://dx.doi.org/10.1007/S10586-019-02911-7/METRICS, URL https://link.springer.com/article/10.1007/s10586-019-02911-7.

[45] R. Medara, R.S. Singh, Energy efficient and reliability aware workflow task scheduling in cloud environment, Wirel. Pers. Commun. 119 (2021) 1301–1320, http://dx.doi.org/10.1007/S11277-021-08263-Z/METRICS, URL https://link.springer.com/article/10.1007/s11277-021-08263-z.

[46] F. Cao, M.M. Zhu, Energy-aware workflow job scheduling for green clouds, in: Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GreenCom-iThings-CPSCom 2013, 2013, pp. 232–239, http://dx.doi.org/10.1109/GREENCOM-ITHINGS-CPSCOM.2013.58.