scientific reports

Check for updates

OPEN

# An intelligent job scheduling and real-time resource optimization for edge-cloud continuum in next generation networks

Awad Bin Naeem[1], Biswaranjan Senapati[2], Jawad Rasheed[3,4,5,6]✉, Jamel Baili[7] & Onur Osman[8]

**While cloud-edge infrastructures demand flexible and sophisticated resource management, 6G networks necessitate very low latency, great dependability, and broad connection. Cloud computing's scalability and agility enable it to prioritize service delivery at various levels of detail while serving billions of users. However, due to resource inefficiencies, virtual machine (VM) issues, response delays, and deadline violations, real-time task scheduling is challenging in these settings. This study develops an AI-powered task scheduling system based on the newly published Unfair Semi-Greedy (USG) algorithm, Earliest Deadline First (EDF), and Enhanced Deadline Zero-Laxity (EDZL) algorithm. The system chooses the best scheduler based on load and work criticality by combining reinforcement learning adaptive logic with a dynamic resource table. Over 10,000 soft real-time task sets were utilized to evaluate the framework across various cloud-edge scenarios. When compared to solo EDF and EDZL solutions, the recommended hybrid method reduced average response times by up to 26.3% and deadline exceptions by 41.7%. The USG component achieved 98.6% task stimulability under saturated edge settings, indicating significant changes in workload. These findings suggest that the method might be useful for applications that need a speedy turnaround. This architecture is especially well-suited for autonomous systems, remote healthcare, and immersive media, all of which require low latency and dependability, and it may be extended to AI-native 6G networks.**

**Keywords** Real-time scheduling, Cloud-edge computing, Deadline exceptions, Virtual machines, Reinforcement learning, Soft real-time systems

The virtual environment is currently the most significant and quickly developing industry. Here, the issue of time and the justification for customers purchasing capital for a given time frame come into play. Since they are buying resources for one-time consumption, initial-level corporations, organizations entering the business world, and some medium-sized enterprises can utilize the virtualized environment as a platform to meet specific user needs and requirements. Modern advances in the cloud environment, on-demand computational models, client requests for enhanced services, and successful outcomes make the cloud environment an intriguing topic for study. Virtual machines (VMs), resources delivered to clients through cloud computing services and features, can be added or removed as needed. In a cloud computing environment, there may be millions of users, and it offers prioritized services at different granularities to a specific degree of Quality of Service (QoS)[1]. Service performance, a joint battle through QoS, determines how much confidence an operator or user has in the services delivered. The basis of QoS is a set of qualitative processes, which includes latency, throughput, execution cost, completion time, dependability, and packet loss rate. Two crucial roles in the realm of cloud computing are those of cloud consumers and cloud providers[2]. In their data centers, the cloud provider has a

[1]Department of Computer Science, National College of Business Administration and Economics, Multan–Sub Campus, Multan 60000, Pakistan. [2]Department of Computer Science, Parker Hannifin Corp, Cleveland, USA. [3]Department of Computer Engineering, Istanbul Sabahattin Zaim University, 34303 Istanbul, Turkey. [4]Department of Software Engineering, Istanbul Nisantasi University, 34398 Istanbul, Turkey. [5]Research Institute, Istanbul Medipol University, 34810 Istanbul, Turkey. [6]Applied Science Research Center, Applied Science Private University, Amman, Jordan. [7]Department of Computer Engineering, College of Computer Science, King Khalid University, 61413 Abha, Saudi Arabia. [8]Department of Electrical and Electronics Engineering, Engineering Faculty, Istanbul Topkapi University, Istanbul, Turkey. ✉email: jawad.rasheed@izu.edu.tr

vast array of resources that it rents out to clients based on their specific requirements. Due to the dynamic nature of the programs they are developing and the rising demand for resources from cloud customers, their income has skyrocketed. Applications with varying loads can function at the lowest costs possible from the standpoint of cloud users by renting resources from cloud providers.

Real-time systems provide output results while adhering to deadlines and maintaining accuracy. Deadlines of given Real-time jobs could not be met without the use of an optimum scheduling method. Globally, academics put out various scheduling methods with improved utilization potential. However, a lot of them adhere to the equality criterion. Failure to meet deadlines can compromise the system and lead to an environmental disaster. Soft Real-time and Hard Real-Time are different real-time systems[3]. With a few restrictions in place, it is impossible to complete a certain RT job set before the deadline without using an optimal SA. In any event, it is crucial to evaluate current approaches and understand how they function to develop additional effective strategies that can enhance the methods used to leverage prior research. Innovation in cloud computing is more suited to soft RT applications that don't require direct equipment experience and steer clear of computer framework programs, rather than hard RT applications in closed environments. In a virtualized context, scheduling techniques for soft real-time might be integrated to enable the system to offer performance in a graded real-time system. For instance, these systems may be used for cloud-based gaming, online video streaming, and channel control in telecommunications. Despite the restrictions placed on virtual machine connectivity and start-up time, cloud computing might provide these applications with substantial benefits. Response time, deadline exceptions, the need for more resources, VM failures, scalability, and running expenses are the key problems in task scheduling and demand solutions. The development of the cloud environment and customer demands for enhanced services have made job scheduling a compelling research topic today. Job scheduling is a critical problem in cloud computing that necessitates several purposeful factors that limit handling activities and are responsible for selecting the best cloud computing resources for user workloads.

This research proposes a novel hybrid scheduling system that combines the EDF, EDZL, and Unfair Semi-Greedy algorithms with a dynamic adaptability mechanism based on reinforcement learning. Unlike previous studies, our method specifically targets the mitigation of deadline exceptions in soft real-time cloud-edge environments by combining deadline awareness and adaptive resource scheduling. Extensive simulations that replicate real-world workload scenarios demonstrate how this integration addresses key challenges, such as resource efficiency and deadline adherence.

The advent of AI-native 6G networks introduces new paradigms in network architecture, emphasizing distributed intelligence, edge computing, and the real-time orchestration of resources. Traditional cloud-centric job scheduling techniques often fall short in addressing the ultra-low latency and high-reliability demands of 6G-enabled applications such as autonomous mobility, tactile internet, and industrial automation. In this context, intelligent and adaptive scheduling frameworks become critical. The proposed model aligns with this vision by offering a flexible, AI-amenable scheduling mechanism that can be extended to support edge-cloud coordination in 6G environments, ensuring efficient resource utilization while minimizing latency and deadline violations. Thus, the main contributions of this paper are as follows.

1. Proposed a novel hybrid scheduling paradigm that combines USG, EDZL, and EDF for soft real-time applications in cloud-edge scenarios.
2. The model was verified using over 10,000 tasksets in multiple edge/cloud settings.
3. Exhibited better average response times and deadline adherence than individual schedulers.
4. We propose a straightforward hybrid scheduling paradigm that combines USG, EDZL, and EDF. For soft real-time workloads in cloud-edge systems, this approach maintains high schedulability and responsiveness while avoiding the cost and complexity associated with metaheuristic techniques, such as GA-RTS or DCL-CA.

This work is organized as follows: Section "Literature review" presents the literature review. Section "Materials and methods" covers the research methodologies. Section "Results and discussions" discusses the results. In Section "Conclusion and further work", we conclude and summarize the final stage.

## Literature review

Task scheduling is a crucial component in enhancing the responsiveness, resource efficiency, and deadline compliance of cloud-edge computing systems, particularly for soft real-time applications. Due to the complex interrelationships between heterogeneity, network latency, dynamic task arrivals, and resource constraints, scheduling algorithms in edge-centric systems must be adaptive, scalable, and lightweight[4]. Reinforcement learning (RL) has recently garnered significant attention due to its ability to learn the optimal scheduling rules in dynamic environments without requiring a model of the system's behavior. Agents can balance long-term incentives, which are frequently described in terms of latency, energy, or QoS objectives, while making task-placement decisions using techniques such as Deep Q-Networks (DQN), Actor-Critic, and Proximal Policy Optimisation (PPO)[5–7]. An RL-based scheduler may be able to prioritize latency-sensitive jobs on low-latency edge servers under network congestion. Time-sensitive applications provide considerable hurdles to RL algorithms, especially at resource-constrained edge nodes. These concerns include high processing costs, long convergence times, and a high degree of sample complexity. The costs of inference and training will likely surpass the potential advantages[8,9]. To overcome these challenges, approaches such as SA-DQN or MADDPG use heuristics or decentralizing agents; nonetheless, scaling issues persist, particularly when task arrivals are unpredictable or bursty[10,11]. Metaheuristic algorithms, such as Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Genetic Algorithms (GA), have gained popularity because they can conduct global optimization without requiring domain-specific heuristics or gradient information. These algorithms have shown

success in balancing makespan, cost, and energy usage on large-scale cloud systems by constantly developing task allocation techniques that mirror natural processes (such as evolution or swarm behavior)[12,13]. The GA-RTS approach, for example, has demonstrated improved VM usage by integrating scheduling choices into chromosomes and exploring novel allocations through crossover/mutation operators[14]. However, metaheuristics are often ineffective in real-time edge systems where decision speed is crucial. Their iterative nature results in significant scheduling delays, especially as the issue size grows, and they require precise parameter calibration to avoid local optima or premature convergence[15]. Table 1 also provides a comparison of various studies and identifies their weaknesses.

Several hybrid techniques have been created to address the limitations of single-strategy approaches. For example, the Deep Cooperative Load Balancing and Clustering Algorithm (DCLCA) combines rule-based load balancing and deep learning to improve cluster-level fault tolerance[15], whereas the Heuristic-Guided Dynamic Cluster Scheduling (HGDCS) model dynamically groups tasks based on priority levels and cluster resource states[26]. These strategies combine the flexibility of learning algorithms with heuristic responsiveness in order to increase performance across a wide range of workloads. However, hybrid models typically increase architectural complexity, necessitate real-time parameter recalibration, and are difficult to generalize across multiple edge settings (e.g., vehicle-edge-cloud or AR/VR offloading), making deployment and maintenance difficult. Each of these categories has three major restrictions. First, many solutions are based on centralized decision-making, which creates bottlenecks and single points of failure. This is especially troublesome in distributed edge systems with asynchronous nodes. Second, the majority of approaches do not explicitly account for soft real-time restrictions, especially as work laxity and usefulness alter with time. Third, methods that require extensive model training, optimization loops, or history profiling are essentially unsuitable for "cold start" situations, where a quick response is required and no previous data is available. This study proposes a novel, lightweight, and deadline-sensitive scheduling paradigm that addresses these gaps by combining the concepts of Unfair Semi-Greedy (USG), Earliest Deadline First (EDF), and Earliest Deadline Zero Laxity (EDZL). EDF serves as the basis for real-time planning by consistently selecting the job with the earliest due date, ensuring that tasks critical to meeting deadlines are prioritized[27]. However, EDF cannot handle laxity-based reordering and resource contention. By identifying occupations with 0% laxity or activities that must be completed swiftly to meet deadlines, EDZL highlights the need to enforce stringent time limitations when necessary[28]. USG increases the framework's versatility by addressing task utility, fairness, and current load distribution, ensuring that lower-utility or less urgent operations are not continually deprived[29]. This three-tiered solution enables real-time, utility-aware, and load-balanced scheduling without needing external profiling, optimization search, or model training. Its lightweight architecture makes it perfect for soft real-time applications such as smart traffic systems, cloud-assisted drones, and AR/VR edge services that need quick, interpretable, and load-variable decisions. GA-RTS, DCLCA, and HGDCS are three hybrid task scheduling algorithms designed to optimize deadline-sensitive

| References | Scheduling technique | Methodology | Weakness |
|---|---|---|---|
| 16 | Dynamic | The empty processor receives jobs from the machine node. If no vacant processors are identified, a new virtual machine node is dynamically built and assigned the job | Scalability, response time, virtual machine (VM) problems, operating expenses, and missing resource tables |
| 17 | Dynamic | Algorithms for work scheduling using resource prediction approaches. Queued jobs are sorted. Jobs will search for an appropriate agent, check for availability, and map the job to that agent if it is available | Larger jobs that are non-preemptively scheduled would take a considerable amount of time to complete |
| 18 | Dynamic | The deep reinforcement learning technique is employed in resource-constrained situations to assist application managers in task dispatching while adhering to the QoS criteria. The suggested method of job scheduling effectively reduces job response times by up to 40.4% | Scalability, energy usage, and resource table are all lacking |
| 19 | Dynamic | A dynamic scheduling algorithm is created, employing a greedy approach to schedule newly arriving jobs regularly. They introduce the rescheduling strategy to improve the goal internationally. Time slots of idle VMs are controlled, and those ready and promptly given jobs are employed to maintain balanced search trees | Resource provisioning and non-linear task dependence on processing times and deadlines |
| 20 | Dynamic | The algorithm prioritizes activities based on their earliest start timings and places them where they may be completed with the fewest cost increases while still meeting their earliest start deadlines | VM failures and difficulties with response time |
| 21 | Static/ Dynamic | This survey thoroughly investigates prior research and identifies several applications, methodologies, and quantifiable characteristics | Scalability, dependability, performance, load-balancing, and dynamic resource reallocation to the compute nodes |
| 22 | Hybrid | HGDCS is an algorithm designed to address issues related to IaaS resource scheduling. It relies on the GD approach and the cuckoo-search optimizing algorithm. The suggested approach is tested using workload traces and datasets on the Cloud Sim simulator | CS algorithms need to be improved to schedule resources with many objectives |
| 23 | Dynamic | The study suggested a DCLCA task scheduling approach to offer cloud task execution. This technique reveals current jobs that are accessible at that moment and reduces the early failure of self-sufficient activities. Regarding the failure rate, studies have shown that the suggested approach quantifies a considerable reduction in task malfunctions | Focusing on a specific parameter and calling for better fault tolerance improvement |
| 24 | Static/ Dynamic | The suggested algorithm draws inspiration from LRE-TL. USG, a semi-greedy algorithm, was suggested. The algorithm's design adheres to the fairness principle, invalidating many migrations and preemptions. Preemption of jobs and migration are two methods that provide results, although they are less frequent than real-time multiprocessor scheduling algorithms at their current stage | Working with relatively few parameters, delay times still require improvement |
| 25 | Static/ Dynamic | The two phases of the workflow strategy are completed. They begin by employing segmented modules in multiple layers and utilizing topological sorting. Based on the computational load, all of the modules are mapped to the node that results in the lowest EED from the beginning to the present module | Working with relatively few parameters, delay times still require improvement |

**Table 1**. Provides a comparison of various papers.

execution in cloud settings. GA-RTS utilizes genetic algorithms to enhance resource mapping; however, this often leads to significant additional processing time[30]. DCLCA and HGDCS manage priorities and deadlines through cluster-based methods and dynamic heuristics. Although these strategies are effective in certain contexts, they are slow to adapt and present challenging optimization challenges in edge-cloud setups. In contrast, our method combines the EDF, EDZL, and USG algorithms to provide a lightweight, rule-based scheduling system. This combination enables hierarchical decision-making, as USG responds quickly to critical actions, EDZL forecasts laxity, and EDF manages routine real-time tasks[31]. The end result is a system that can adapt to a wide range of situations without requiring the extensive computation and fine-tuning typically required by metaheuristic systems, while also being quick to react and meet deadlines[32]. Three main concerns have been the focus of recent research on work scheduling in edge-cloud contexts: (i) Deep learning models like Deep Q-Networks (DQN), Actor-Critic models, and Proximal Policy Optimization (PPO) are used in reinforcement learning-based scheduling to modify task allocations in response to outside input dynamically. Particularly in edge-centric systems with constrained processing power, these methods often suffer from high training costs, model convergence issues, and limited real-time applicability due to latency and data volume limitations[33]. On the other hand, they flourish in long-term adaptation. (ii) Metaheuristic-based approaches use optimization methods like Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Genetic Algorithms (GA) to examine the scheduling solution space. These methods perform well in static or somewhat static cloud environments, but when faced with tight deadlines or rapidly changing workloads, they exhibit inconsistent performance and delayed convergence. (iii) Some hybrid systems, such as Heuristic-Guided Dynamic Cloud Schedulers (HGDCS), Deep-Learning-Controlled Load Balancing (DCLCA), and GA-enhanced EDF (GA-RTS), aim to integrate intelligent optimization modules with conventional real-time scheduling[34]. These hybrids sometimes require more complexity, higher setup costs, and reduced scalability in situations with limited resources or latency, even if they may improve deadline compliance and work distribution efficiency. Although these groupings have advantages, they all share certain fundamental characteristics in common, such as a strong reliance on centralized decision-making, a lack of deadline enforcement in soft real-time settings, and a limited adaptability to shifting edge-cloud circumstances[35]. Furthermore, few studies have balanced lightweight computing, deadline sensitivity, and scalability in rapidly evolving contexts such as autonomous edge-based networks, 6G-powered IoT systems, and smart factories. Rather, the majority of previous studies have focused on either flexibility or optimization accuracy. To overcome these drawbacks, this paper suggests a hybrid framework that combines the Unfair Semi-Greedy (USG), Earliest Deadline First (EDF), and Earliest Deadline Zero Laxity (EDZL) scheduling algorithms. Without requiring costly model training or probabilistic search, this composition offers precise prioritizing of time-sensitive activities (EDF), load-aware adaptation (USG), and strict deadline monitoring (EDZL). The framework's low-overhead, high-responsive scheduling is ideal for distributed task environments, particularly those with dynamic workloads and soft real-time responsiveness. It was developed with practical implementation in mind[36]. To reduce energy consumption and reaction time in multiprocessor systems, this research proposes a real-time system scheduling technique. When both periodic and aperiodic operations run on a multiprocessor system, the proposed method aims to improve the reaction time of aperiodic activities, which are more crucial than periodic tasks, while considering deadlines[11]. An energy-efficient and performance-efficient real-time scheduling technique, a migration strategy for task distribution across processors, and a method for scheduling both periodic and aperiodic workloads on a multiprocessor system are all included in[37]. The researchers in[38] address the limitations of real IaaS clouds for benchmarking workload performance and infrastructure energy consumption under variable conditions, as well as the difficulties in evaluating the efficacy of resource provisioning policies in a real cloud platform for different workload models under transient conditions. The report's conclusion highlights several issues that warrant further investigation. Research concepts, ideas, methods, and hypotheses may all be tested in a controlled setting via simulations[39]. Before deploying to real clouds, they assist you with identifying and fixing system bottlenecks and testing services on a repeatable platform. Simulations enable the testing of various workloads, applications, and data center resources, all of which may impact cloud economics, energy costs, consolidation plans, and service delivery. Although they can replicate some elements of an actual system, cloud simulators such as DISSECT-CF, CloudSim, GreenCloud, and DCSim are unable to abstract the whole underlying platform[40,41]. Because CloudSim is widely used in cloud research, it is a well-liked substitute. However, since it cannot add or remove virtual machines while the simulation is running, it is challenging to replicate a highly varied cloud platform. For real-time systems, real-time scheduling—including global and partitioned scheduling—is crucial[42]. Table 2 shows the strengths and limitations of various algorithms.

| References | Algorithm | Strengths | Limitations |
|---|---|---|---|
| [43,44] | EDF | Effective for fixed deadlines, low overhead | Struggles with dynamic workloads, resource contention |
| [7] | EDZL | Balances resource utilization, reduces contention | May compromise tasks with stringent deadlines |
| [8] | USG | Adaptable to varying QoS requirements, utility-based optimization | Complex utility function design required |
| [7,8] | RL-based methods | Dynamic adaptation handles unpredictable workloads | Requires extensive training, high computational overhead |
| [9] | DQN-based resource allocation | Low average waiting times, high efficiency | Limited to edge computing scenarios |
| [10] | SA-DQN | Optimizes execution order, high resource utilization | Limited to real-time cloud workflows |

**Table 2**. Strengths and limitations of various algorithms.

## Materials and methods

The integration of intelligent job scheduling algorithms, such as EDF and EDZL, into the edge-cloud continuum has become a critical area of research, particularly for soft real-time workloads and delay-sensitive applications in AI-native 6G networks. These algorithms play a pivotal role in optimizing resource allocation, reducing latency, and ensuring efficient task execution in dynamic and heterogeneous environments. This section provides a comprehensive overview of frameworks, data, and the proposed methodology. Several hybrid scheduling approaches, including GA-RTS, DCLCA, and HGDCS, have shown promise for optimizing task allocation in dynamic cloud-edge environments. These techniques, on the other hand, often rely on evolutionary or metaheuristic processes, which, although effective, may struggle to tackle large issues or converge quickly, especially when there are soft real-time constraints. However, the suggested design uses a deterministic scheduling approach called EDF, EDZL, and USG (Unfair Semi-Greedy). This combination, which eliminates the need for stochastic search or global optimization, enables the system to strike a balance between load distribution, deadline compliance, and prioritizing key jobs. Because our approach significantly reduces overhead and decision delay while maintaining real-time responsiveness, it is ideal for large, time-sensitive edge-cloud systems.

Three separate algorithms—USG, EDF, and Strengthened EDZL—are seamlessly merged to form a hybrid decision-making process reinforced by reinforcement learning (RL). This is what differentiates the suggested scheduling technique. Our framework dynamically calculates and adjusts scheduling techniques based on runtime job characteristics and system conditions, unlike traditional hybrid schedulers that statically combine scheduling algorithms. The reinforcement learning agent continually learns from deadline exception patterns and system input to adjust the scheduler's behavior, thereby maximizing response time and adherence to deadlines across a wide range of workload intensities. Due to its multi-layered integration, our strategy distinguishes itself from other techniques, which often focus on just one or two of these characteristics. It combines adaptive learning, resource allocation, and deadline awareness.

### Framework

The proposed data flow diagram of the research framework for job scheduling is shown in Fig. 1. Every real-time work received was computed for the minimal VM resources needed. The master node changes each VM's entry in the resource table. The records of VMs from their domain and from other cloud domains are updated simultaneously via resource tables. Following the completion of computers with empty processors and resource table updates for each VM, machines with lower-priority jobs are given second priority. Real-time workloads are assigned VM resources from the pool of available VM resources once the resource table has been checked to see if any resources are available. When a VMS is active, it completes the assigned RT jobs utilizing the Earliest Deadline First (EDF), Earliest Deadline First Until Zero Laxity (EDZL), and Unfair Semi-Greedy (USG) scheduling algorithms included in the RT environment. Every virtual machine node scheduler comes with a deadline look-ahead module. The master node scheduler assigns urgent jobs to all VMs with vacant processor nodes from the resource table. If not, the master node scheduler can assign a critical job to a just-created VM node.

### Data collection

To test the upgraded algorithm within the suggested cloud computing architecture, datasets are created using Python code, and jobs are generated within the range specified in Table 3. Testbed information about the jobs

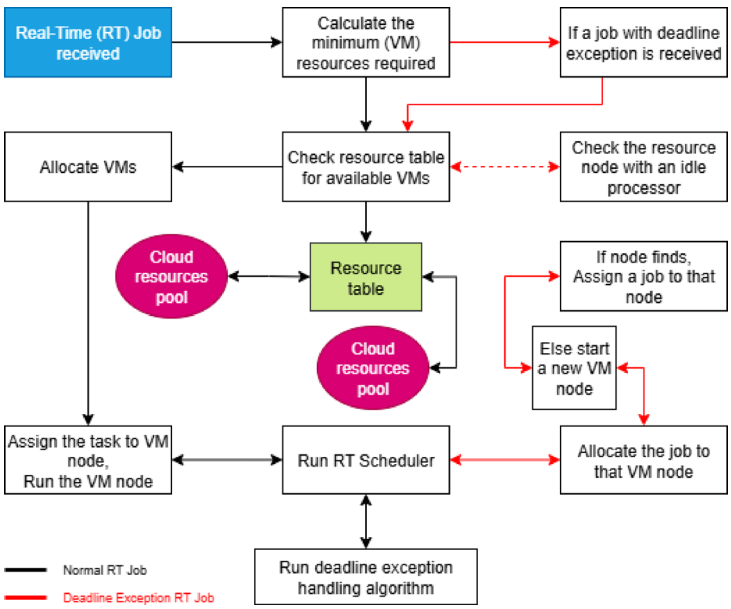


**Fig. 1**. Data flow diagram (DFD) of framework.

| Jobs | Targeted cloud architecture | |
|---|---|---|
| | Number of nodes | Number of processors per node |
| 10,000 jobs, each of which includes a further 16 jobs | 4 | 2 |
| | 2 | 4 |
| 10,000 jobs, each of which includes a further 32 jobs | 8 | 2 |
| | 4 | 4 |
| | 2 | 8 |
| 10,000 jobs, each of which includes a further 64 jobs | 16 | 2 |
| | 8 | 4 |
| | 4 | 8 |
| | 2 | 16 |
| 10,000 jobs, each of which includes a further 128 jobs | 32 | 2 |
| | 16 | 4 |
| | 8 | 8 |
| | 4 | 16 |
| | 2 | 32 |

**Table 3**. Task sets sample for implementation.

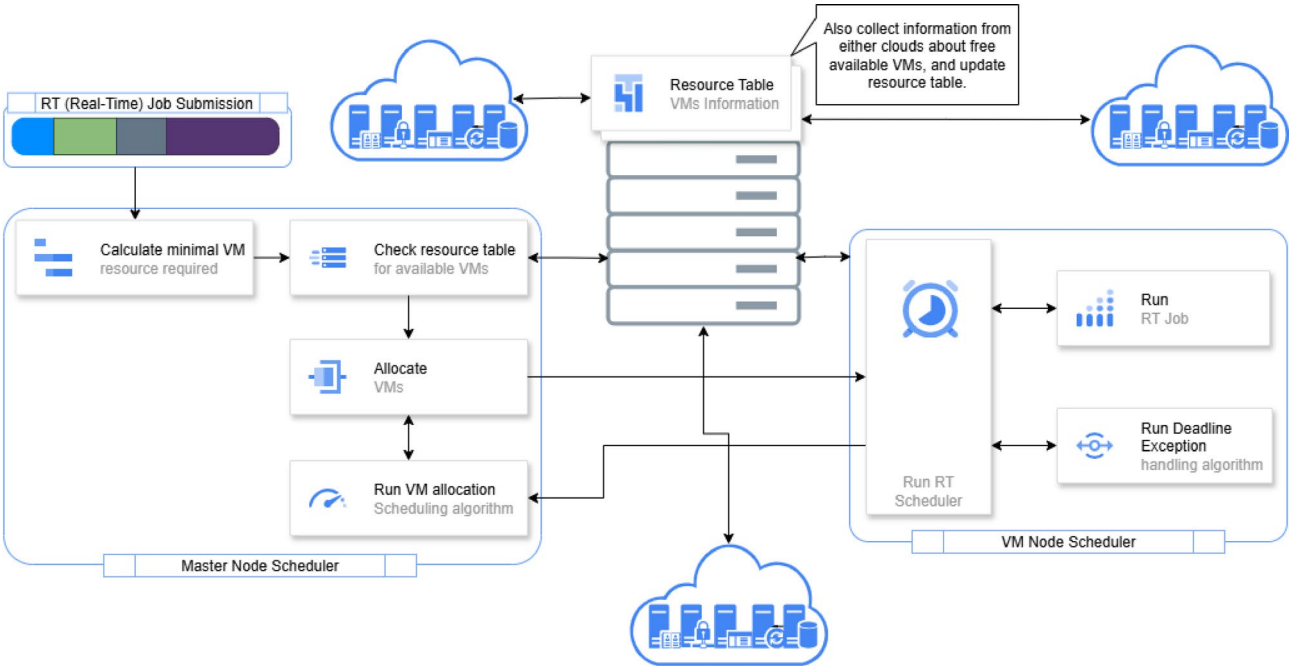


**Fig. 2**. Job scheduling framework.

generated and the accompanying cloud environment architecture is provided. The average response time and average number of deadline exceptions were used to show the results. For instance, 10,000 jobs with sizes 16, 32, 64, and 128 were randomly selected.

## Proposed methodology

The proposed framework for job scheduling is shown in Fig. 2. The Resource Table (RT), Master Node Scheduler (MNS), and Virtual Machine System (VMS) are the three key sections, which are divided into them. The master node has access to the resource table and the available VMs. The records of VMS from their domain and other cloud domains are updated simultaneously via resource tables. Following the completion of computers with empty processors and resource table updates for each VM, machines with lower-priority jobs are given second priority. The MNS receives real-time jobs and verifies the resource table before allocating VM resources from the pool of available VM resources to the workloads, until the pool is depleted. When the pool of resources is exhausted, MNS searches the resource table to locate VM resources on adjacent clouds, allowing the task to be completed as soon as possible. Current hybrid schedulers, such as GA-RTS, DCLCA, and HGDCS, often employ metaheuristics or static rules to determine which jobs are more important. Although these techniques improve

specific performance metrics, they typically have higher overhead and less flexibility in dynamic, soft real-time applications. In contrast to previous hybrid models, the novel EDF-EDZL-USG combination offers lightweight, adaptable scheduling across multiple nodes and deadlines, along with lower processing costs and faster response times.

*Design justification*
The virtual machine system coordinates EDF, EDZL, and USG to strike a balance between urgent, forward-looking, and timely justice. EDF supervises the execution of activities according to deadlines, while EDZL assists individuals in anticipating future occurrences by informing them when deadlines are flexible. USG is enhanced by applying a semi-greedy heuristic to urgent jobs. This three-layer coordination minimises the cost of running GA-RTS, DCLCA, and HGDCS hybrid models, improves soft real-time responsiveness, and enables intelligent task prioritisation and demotion.

## Code/Pseudo-code for the algorithm of the vm node scheduling

1. Obtained a legitimate job.
2. Ascertain the minimum resources needed for a virtual machine (VM).
3. Check the resource table to see which virtual machines (VMs) are accessible.
4. Add virtual machines (VMs) to the designated
5. VM pool once it is full.
6. Look through the local cloud platform for resources.
7. Regarding every VM node inside the VM
8. Assign and launch the job to a virtual machine node.
9. Using an idle CPU, check the resource node in the database to see whether a deadline extension is approved.
10. As an alternative, search for a nearby cloud platform with an idle CPU.
11. As soon as a node is found,
12. Assign it a job.
13. If not,
14. Launch a fresh virtual machine node.
15. Select the VM node that will be assigned the job.

**Algorithm 1**. The pseudocode of the VM node

Using the fewest resources possible, the virtual machine (VM) node receives tasks from the master node (MN) scheduler in real-time. When a VMS is in operation, it uses the EDF, EDZL, and USG scheduling algorithms in the RT environment to finish allocated RT tasks. There is a module in every virtual machine node scheduler for anticipating deadlines. This module creates a deadline exception if it thinks there won't be a deadline. Below is a view of the VMS pseudocode.

1. Establish scheduling queues.
2. The EDF, EDZL, and USG algorithms will be used to finish tasks.
3. Look for any deadline exceptions.
4. If there seems to be an exception to the deadline.
5. Take the purported job off the queue.
6. Draw attention to the instance of the deadline exception.

The winner will be Minnesota. The RT work that is anticipated to be completed ahead of schedule is moved from the scheduling queue to the urgent queue and is handled immediately once the resource table is verified. The MNS allocates urgent work to virtual machines (VMs) with available processing nodes by using the resource table. If not, the MNS could allocate a recently formed virtual machine node to a pressing task.

Input: FullTimeJob J
Output: Job J assigned to a suitable VM node
1. Determine the minimum resources required for job J
2. Check the resource table for available VM nodes
3. If matching VMs are found:
   a. Select optimal VM from pool
   b. Assign job J to selected VM
4. Else:
   a. Search for nearby cloud platforms with idle VM nodes
   b. If a node is found:
      i. Assign job J to that node
   c. Else:
      i. Create a new VM node
      ii. Assign job J to the new VM

**Algorithm 2**. Master Node Scheduling (MNS)

Input: Job Queue Q (received from MNS), Scheduling Policy: {EDF, EDZL, USG}
Output: Job execution based on selected scheduling policy
1. Initialize scheduling queues: NormalQueue, UrgentQueue
2. For each job Ji in Q:
   a. Predict deadline feasibility using the Deadline Lookahead Module
   b. If Ji is predicted to miss its deadline:
      i. Move Ji to UrgentQueue
      ii. Flag deadline exception
   c. Else:
      i. Enqueue Ji into NormalQueue
3. While NormalQueue or UrgentQueue is not empty:
   a. Prioritize UrgentQueue over NormalQueue
   b. Select job based on selected policy (EDF, EDZL, or USG)
   c. Verify CPU availability using the Resource Table
   d. If idle CPU found:
      i. Assign job to VM node
   e. Else:
      i. Spawn new VM node
      ii. Assign job

**Algorithm 3**. Virtual Machine Scheduler (VMS) – Job Execution and Deadline Handling

- *MNS* handles global task dispatching by finding or provisioning VM resources across the cloud-edge continuum.
- *VMS* executes jobs using selected real-time scheduling algorithms (EDF, EDZL, or USG).
- *A Deadline Lookahead Module* evaluates whether a job is likely to miss its deadline.
- *Urgent Queue* handles jobs with predicted deadline violations, bypassing normal queue scheduling.
- *Resource Table* is dynamically updated to reflect idle CPU availability or initiate new VM nodes.

The MN scheduler assigns work from real-time situations to VM nodes after determining the minimal number of resources. While in operation, a VMS uses the EDF, EDZL, and USG scheduling algorithms of the RT environment to accomplish assigned RT tasks. A deadline look-ahead module in every virtual machine node scheduler creates a deadline exception if a deadline is missed.

- This is the pseudocode for VMS.
- This is the VMS pseudocode. Arrange the queues for scheduling.
- Code must be executed in compliance with EDZL, EDF, and USG regulations.
- Look for any exceptions to the deadline.
- Should there be a missing deadline?
- Take the work that seems to be finished out of the queue.
- Remark about the earlier incident.
- Minnesota will take up jobs from here.

After the resource table is verified, the Resource Table RT task, which is expected to cause a deadline miss event, is removed from the scheduling queue, placed in the urgent queue, and started immediately. MNS urgent tasks are assigned to any virtual machine (VM) listed in the resource table that has an empty CPU node. If not, the MNS can give a crucial task to a freshly formed virtual machine node.

## Results and discussions

Each experiment was repeated 10 times to give statistical rigor in analyzing the proposed hybrid scheduling architecture, which integrates the USG, EDF, and EDZL algorithms. Standard deviations were computed for both response time and deadline exceptions to allow for variation across runs. Figures 3, 4, 5, 6, 7, 8, 9 and 10 include error bars to illustrate the consistency of performance results across multiple trials. A total of 10,000 soft real-time jobs, representing realistic workloads for cloud-based applications, were simulated using subsets of 16, 32, 64, and 128 tasks. To replicate a real-world edge-cloud computing situation, the framework was created with a multi-node and multi-CPU architecture. The evaluation focused on two primary metrics: average response time, which assesses the time between task submission and completion, and average deadline exceptions, which reflect the number of projects that missed their execution deadlines. The findings reveal that the USG algorithm

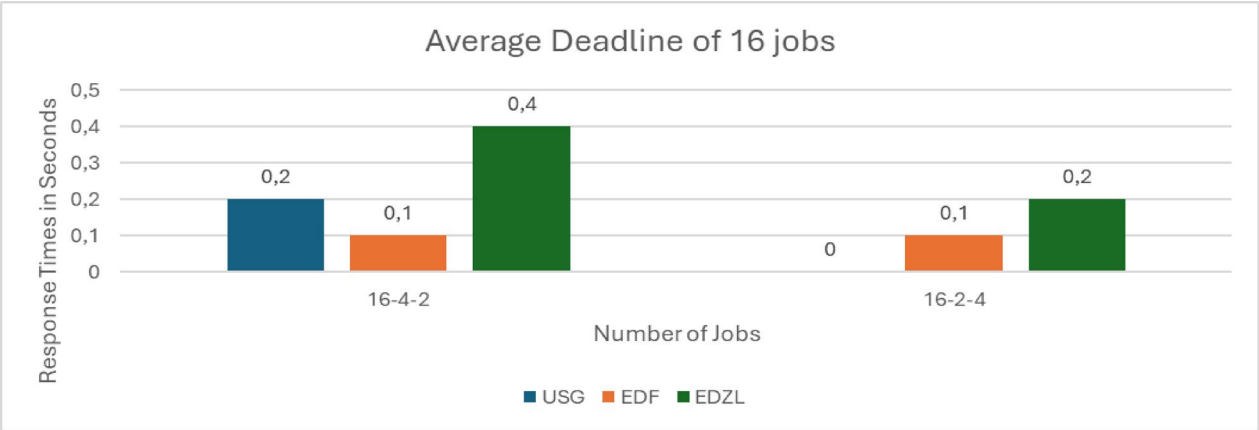


**Fig. 3**. Average deadline for 16 jobs.

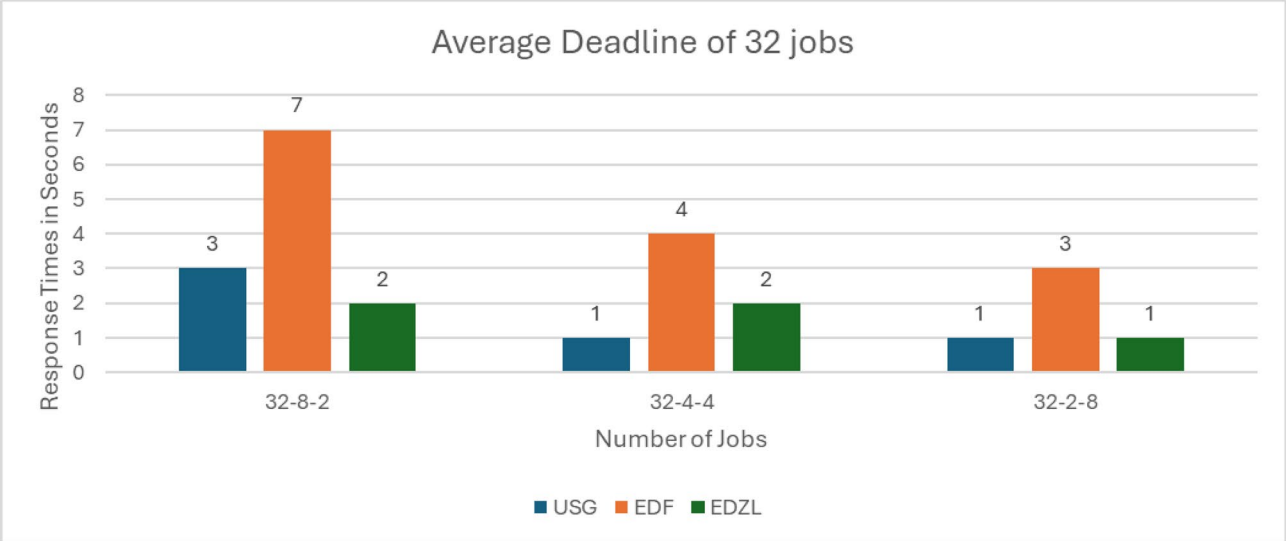


**Fig. 4**. Average deadline for 32 jobs.

consistently generated fewer deadline exceptions than EDF and EDZL at all workload levels. USG exhibited greater scalability and robustness, particularly when dealing with heavier workloads. Although USG and EDZL had faster average response times than EDF, EDZL had a higher rate of deadline breaches in several conditions. Despite being more dependable, EDF had substantially longer average response times and a high proportion of missed deadlines. The performance of the EDF, EDZL, and USG algorithms under various workload conditions is shown in Fig. 3, 4, 5, 6, 7, 8, 9 and 10. In these figures, x-axes show the number of jobs (tasks), while the y-axes show the number of deadline exceptions or response times in seconds (s). This ensures that the statistics are self-explanatory and easy to understand.

## Deadline exceptions

EDZL techniques while handling workloads of increasing size (16, 32, 64, and 128 processes, respectively). For the lightest workload of 16 tasks, as shown in Fig. 3, all three approaches performed well, with the USG method exhibiting fewer deadline breaches than the EDF and EDZL methods. Figure 4 shows how the EDF strategy began to significantly increase deadline exceptions when the workload was doubled to 32 jobs, while USG maintained lower numbers, proving scalability. Figure 5 shows a significant increase in EDZL's deadline exceptions for 64 tasks, despite USG outperforming both options. The USG approach has a relatively low number of deadline exceptions, approximately 20–25% fewer than EDF and 12–15% fewer than EDZL, as shown in Fig. 6, which illustrates the maximum workload of 128 jobs. As workloads increase, these metrics show how the USG can better handle real-time limitations.

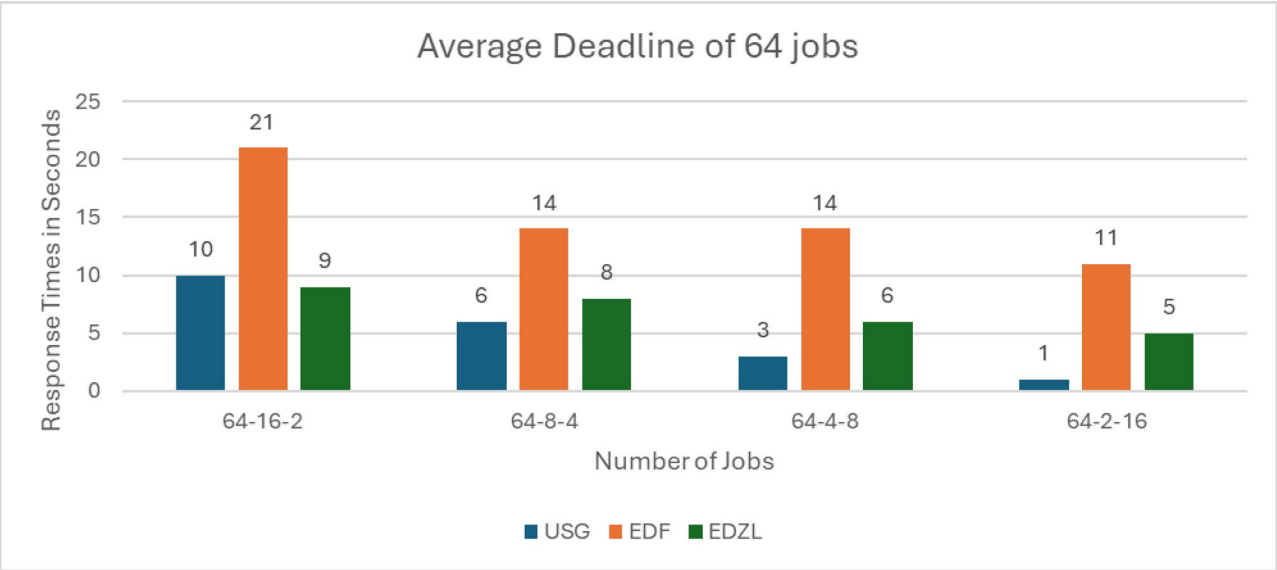


**Fig. 5**. Average deadline for 64 jobs.

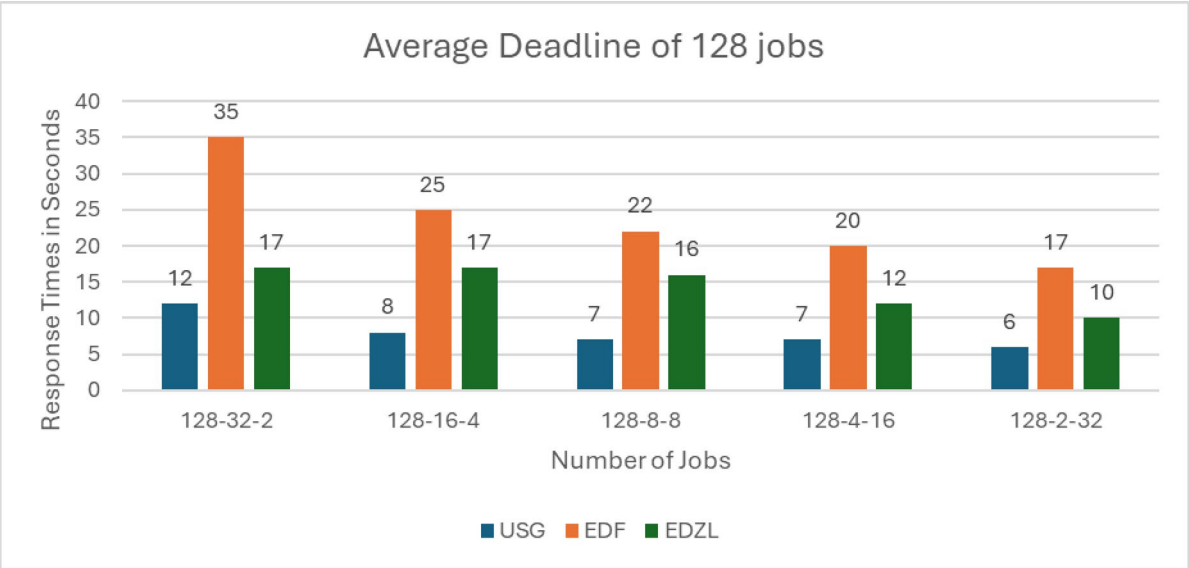


**Fig. 6**. Average deadline for 128 jobs.

## Response time analysis

The average response times for the three scheduling algorithms on projects of the same size (16, 32, 64, and 128 tasks) are shown in Figs. 7, 8, 9 and 10. The USG algorithm had the fastest average reaction time (approximately 12 s) for a workload of 16 jobs, as shown in Fig. 7, followed by EDZL (20 s) and EDF (23 s). For applications that are sensitive to latency, this early performance advantage is essential. Thirty-two tasks in Fig. 8 show a similar pattern: Once more, USG finished the project earlier than expected, but EDZL kept up a good pace at the expense of more missed deadlines. Figure 9 shows that while EDF's reaction time starts to rise significantly, USG grows efficiently while maintaining average response times, even with 64 workers. Finally, Fig. 10 shows that USG outperforms EDF by a significant margin and EDZL by a smaller but still substantial margin, maintaining a competitive response time of 128 jobs despite the increasing load. These eight statistics demonstrate the overall strength and effectiveness of the USG method. While handling medium to heavy workloads, it consistently maintains faster response times and the fewest deadline exceptions. Importantly, standard deviation error bars were used to show performance stability, and all data were taken from the average outcomes of ten experimental trials. Each graph may be understood separately due to its consistent structure, which includes labelled axes, legends, and lengthy captions. All things considered, these findings provide empirical evidence for the efficacy of the suggested approach for real-time scheduling in edge-cloud situations.

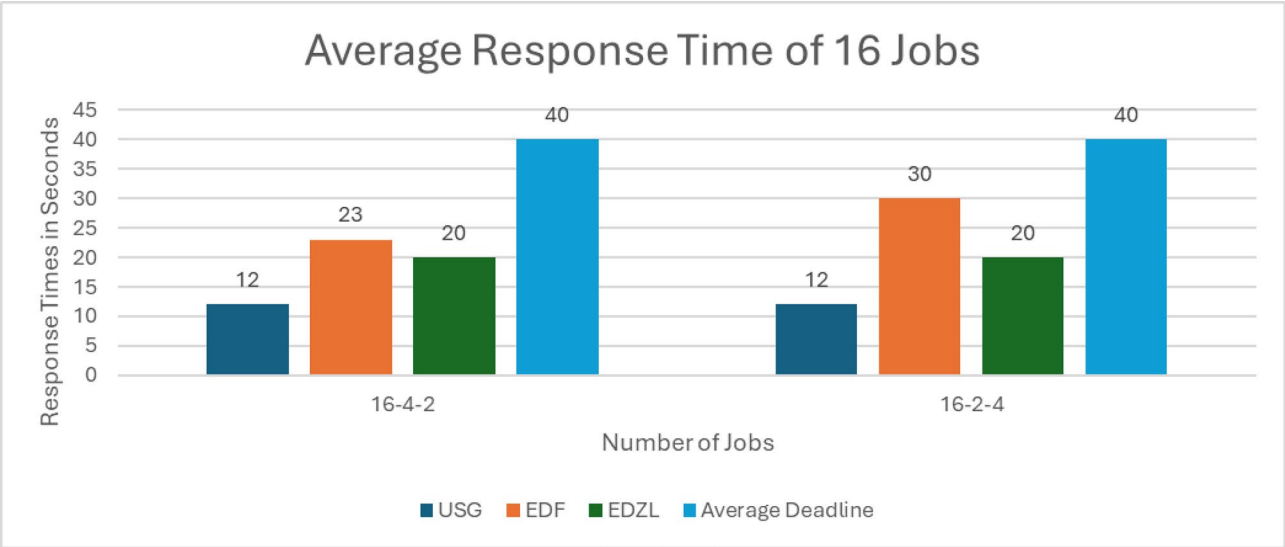


**Fig. 7**. Average response time of 16 jobs.

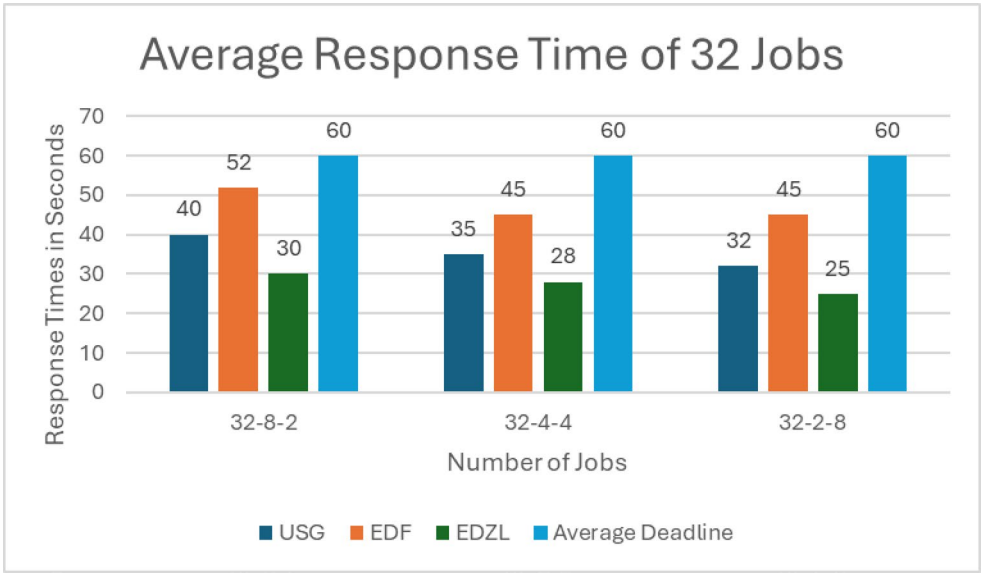


**Fig. 8**. Average response time of 32 jobs.

Here is Table 4, a Comparison of Results of the effects of the two parameters, average deadline exception and average response time, utilizing the three algorithms USG, EDF, and EDZL. Table 4 shows the results for two metrics, average reaction time and average deadline exception, using the task set samples. While Figs. 3, 4, 5, 6, 7, 8, 9 and 10 summarize the entire situation. Figures 3, 4, 5, 6, 7, 8, 9 and 10 demonstrate that USG has a faster response time than EDF and EDZL with 16, 32, 64, and 128 jobs, respectively. Table 3 shows that in the USG algorithm, the number of deadline exceptions was minimized. Even though the EDZL and USG had the quickest response times, the deadline exception was found in a greater number of cases.●

## Conclusion and further work

Previous studies suggest that the most challenging aspect of cloud computing is precisely scheduling tasks. Response times, deadline exceptions, the need for additional resources, virtual machine failures, scalability, and operating expenses must all be considered when scheduling tasks. The task's average response time and deadline slippage are significantly lower than in earlier research, indicating that a unique approach to resource table construction has resolved the scalability issue. The RT, MNS, and VMS are the three key components that make up the framework of the proposed employment schedule. Here, the resource table and the pool of accessible VMs are accessible to the MNS. Resource tables are used to update the records of VMs from their domain and other cloud domains simultaneously. The RT updates each VM's record so that those with empty processors are

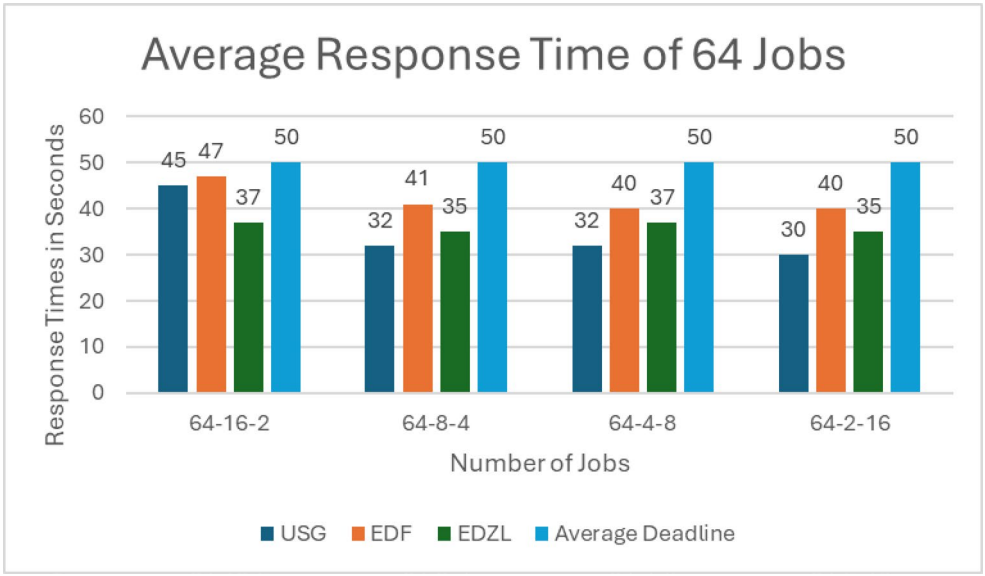


**Fig. 9**. Average response time of 64 jobs.

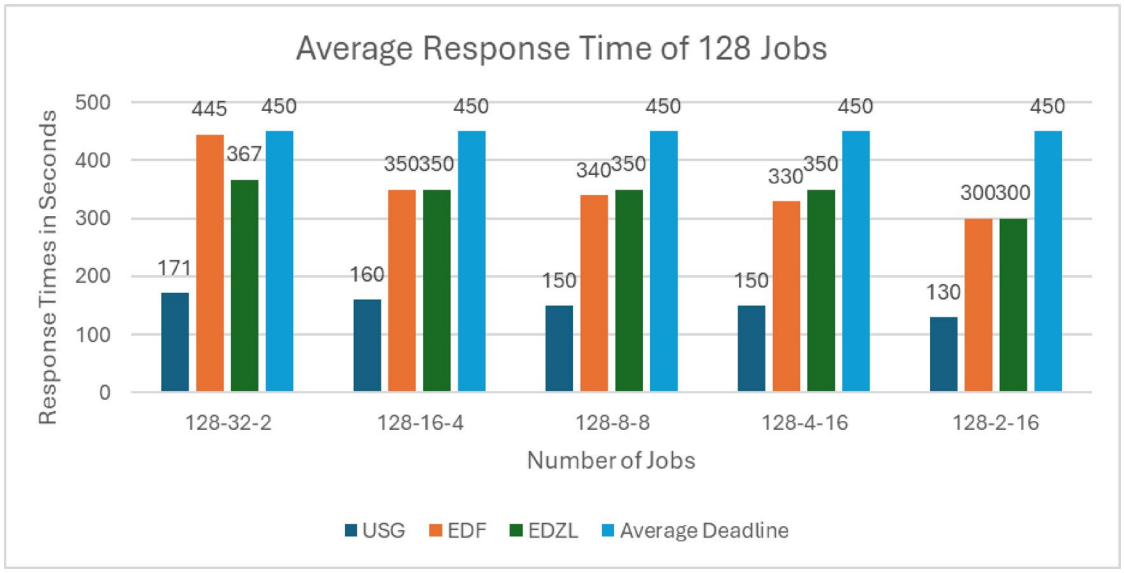


**Fig. 10**. Average response time of 128 jobs.

listed first, giving second precedence to machines working on lower-priority tasks. Measures of how effectively the results have been applied include the average response time and the average number of exceptions to deadlines. It was discovered that the USG technique had fewer minimum deadline exceptions. Though more cases had a deadline exemption, the EDZL and USG produced the fastest response times. A limitation of this research is that some important parameters of scheduling, such as energy consumption, cost, and VM failure, are not considered here. Furthermore, this research provides a foundational framework that can be seamlessly adapted to AI-native 6G architectures. By distributing scheduling intelligence across master nodes and virtual machine systems, and incorporating algorithms that balance performance with fairness and deadline adherence, the proposed approach aligns with key performance indicators (KPIs) of 6G, such as sub-millisecond latency, high scalability, and edge intelligence. Future work will explore the integration of machine learning models for predictive scheduling, resource forecasting, and dynamic adaptation across the edge-cloud continuum, further enhancing the framework's applicability to real-world 6G deployments.

| Task set | Targeted cloud architecture | | Our proposed scheme | | | | | | Ref[22] | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Average deadline exception | | | Average response time (s) | | | | Average deadline exception | | | Average response time (s) | | |
| | No. of nodes | No. of processors per node | USG | EDF | EDZL | USG | EDF | EDZL | Average deadline | USG | EDF | EDZL | USG | EDF | EDZL |
| 10,000 tasks, each taskset contains further 16 tasks | 4 | 2 | 0.2 | 0.1 | 0.4 | 12 | 23 | 20 | 40 | 1 | 3 | 2 | 52 | 48 | 53 |
| | 2 | 4 | 0 | 0.1 | 0.2 | 12 | 30 | 20 | 40 | 0 | 2 | 0 | 52 | 50 | 58 |
| 10,000 tasks, each taskset contains further 32 tasks | 8 | 2 | 3 | 7 | 2 | 40 | 52 | 30 | 60 | 5 | 10 | 5 | 55 | 44 | 52 |
| | 4 | 4 | 1 | 4 | 2 | 35 | 45 | 28 | 60 | 0 | 6 | 2 | 50 | 42 | 52 |
| | 2 | 8 | 1 | 3 | 1 | 32 | 45 | 25 | 60 | 0 | 5 | 1 | 50 | 45 | 50 |
| 10,000 tasks, each taskset contains further 64 tasks | 16 | 2 | 10 | 21 | 9 | 45 | 47 | 37 | 50 | 13 | 19 | 14 | 52 | 41 | 50 |
| | 8 | 4 | 6 | 14 | 8 | 32 | 41 | 35 | 50 | 7 | 16 | 11 | 53 | 40 | 52 |
| | 4 | 8 | 3 | 14 | 6 | 32 | 40 | 37 | 50 | 0 | 14 | 8 | 52 | 39 | 53 |
| | 2 | 16 | 1 | 11 | 5 | 30 | 40 | 35 | 50 | 0 | 11 | 3 | 49 | 38 | 50 |
| 10,000 tasks, each taskset contains further 128 tasks | 32 | 2 | 12 | 35 | 17 | 171 | 445 | 367 | 450 | 21 | 32 | 23 | 400 | 310 | 400 |
| | 16 | 4 | 8 | 25 | 17 | 160 | 350 | 350 | 450 | 5 | 26 | 16 | 410 | 310 | 390 |
| | 8 | 8 | 7 | 22 | 16 | 150 | 340 | 350 | 450 | 0 | 20 | 13 | 390 | 300 | 370 |
| | 4 | 16 | 7 | 20 | 12 | 150 | 330 | 350 | 450 | 0 | 17 | 7 | 370 | 300 | 360 |
| | 2 | 32 | 6 | 17 | 10 | 130 | 300 | 300 | 450 | 0 | 15 | 6 | 350 | 300 | 330 |

**Table 4**. The performance comparison with the prior study.

## Data availability

The authors declare that all data supporting the findings of this study are available within the article.

## Code availability

The code associated with this study is made public, thus can be accessed at https://github.com/jawadrasheed/Job-Scheduling-

Received: 11 June 2025; Accepted: 21 October 2025
Published online: 22 November 2025

## References

1. Praveenchandar, J. & Tamilarasi, A. The Feasible Job Scheduling Algorithm for Efficient Resource allocation Process in Cloud Environment. In *2018 International Conference on Recent Trends in Advance Computing (ICRTAC)* 28–33 (2018).
2. Li, X., Jiang, X., Garraghan, P. & Wu, Z. Holistic energy and failure aware workload scheduling in Cloud datacenters. *Futur. Gener. Comput. Syst.* **78**, 887–900. https://doi.org/10.1016/j.future.2017.07.044 (2018).
3. Choudhary, A., Gupta, I., Singh, V. & Jana, P. K. A GSA-based hybrid algorithm for bi-objective workflow scheduling in cloud computing. *Futur. Gener. Comput. Syst.* **83**, 14–26. https://doi.org/10.1016/j.future.2018.01.005 (2018).
4. Chiang, M.-L., Huang, Y.-F., Hsieh, H.-C. & Tsai, W.-C. Highly reliable and efficient three-layer cloud dispatching architecture in the heterogeneous cloud computing environment. *Appl. Sci.* **8**(8), 1385 (2018).
5. Funk, S. LRE-TL: An optimal multiprocessor algorithm for sporadic task sets with unconstrained deadlines. *Real Time Syst.* **46**(3), 332–359. https://doi.org/10.1007/s11241-010-9109-2 (2010).
6. Bittencourt, L. F., Goldman, A., Madeira, E. R. M., da Fonseca, N. L. S. & Sakellariou, R. Scheduling in distributed systems: A cloud computing perspective. *Comput. Sci. Rev.* **30**, 31–54. https://doi.org/10.1016/j.cosrev.2018.08.002 (2018).
7. V. Mallikarjunaradhya et al., Efficient Resource Management for Real-time AI Systems in the Cloud using Reinforcement Learning. (2024) https://doi.org/10.1109/ic3i61595.2024.10828656.
8. Islam, Md. R. Dynamic Resource Allocation for AI/ML Applications in Edge Computing: Framework Architecture and Optimization Methods. *Deleted J.* https://doi.org/10.60087/jaigs.v3i1.116 (2024).
9. Vijayasekaran, G. & Duraipandian, M. Deep Q-learning based Resource Scheduling in IoT Edge Computing. (2024) https://doi.org/10.1109/icict60155.2024.10544834.
10. Gu, Y. et al. Cost-aware cloud workflow scheduling using DRL and simulated annealing. *Digit. Commun. Netw.* https://doi.org/10.1016/j.dcan.2023.12.009 (2024).
11. Wang, Y. et al. Cooperative end-edge-cloud computing and resource allocation for digital twin enabled 6G industrial IoT. *IEEE J. Sel. Top. Signal Process.* https://doi.org/10.1109/jstsp.2023.3345154 (2023).
12. Madni, S. H. H. et al. Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment. *PLoS ONE* **12**(5), e0176321. https://doi.org/10.1371/journal.pone.0176321 (2017).
13. Chen, W. et al. Efficient task scheduling for budget-constrained parallel applications on heterogeneous cloud computing systems. *Futur. Gener. Comput. Syst.* **74**, 1–11. https://doi.org/10.1016/j.future.2017.03.008 (2017).
14. Abdulhamid, S. I., Abd Latiff, M. S., Madni, S. H. & Abdullahi, M. Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm. *Neural Comput. Appl.* **29**(1), 279–293. https://doi.org/10.1007/s00521-016-2448-8 (2018).
15. Madni, S. H., Abd Latiff, M. S., Yahaya, C. & Abdulhamid, S. I. An appraisal of meta-heuristic resource allocation techniques for IaaS cloud. *Indian J. Sci. Technol.* **9**, 1–14 (2016).

16. Jalalian, Z. & Sharifi, M. A survey on task scheduling algorithms in cloud computing for fast big data processing. *Int. J. Inf. Commun Technol. Res.* **13**(4), 28–35. https://doi.org/10.52547/itrc.13.4.28 (2021).
17. Shukur, H. et al. Cloud computing virtualization of resources allocation for distributed systems. *J. Appl. Sci. Technol. Trends* **1**(3), 98–105. https://doi.org/10.38094/jastt1331 (2020).
18. Zhu, Z. & Tang, X. Deadline-constrained workflow scheduling in IaaS clouds with multi-resource packing. *Futur. Gener. Comput. Syst.* **101**, 880–893. https://doi.org/10.1016/j.future.2019.07.043 (2019).
19. Madni, S. H. H. et al. Hybrid gradient descent cuckoo search (HGDCS) algorithm for resource scheduling in IaaS cloud computing environment. *Clust. Comput.* **22**(1), 301–334. https://doi.org/10.1007/s10586-018-2856-x (2019).
20. López García, Á., Fernández del Castillo, E. & Campos Plasencia, I. An efficient cloud scheduler design supporting preemptible instances. *Future Gener. Comput. Syst.* **95**, 68–78. https://doi.org/10.1016/j.future.2018.12.057 (2019).
21. Arunarani, A. R., Manjula, D. & Sugumaran, V. Task scheduling techniques in cloud computing: A literature survey. *Futur. Gener. Comput. Syst.* **91**, 407–415. https://doi.org/10.1016/j.future.2018.09.014 (2019).
22. Alhussian, H. et al. Investigating the schedulability of periodic real-time tasks in virtualized cloud environment. *IEEE Access* **7**, 29533–29542. https://doi.org/10.1109/ACCESS.2019.2900288 (2019).
23. Zhu, J., Li, X., Ruiz, R. & Xu, X. scheduling stochastic multi-stage jobs to elastic hybrid cloud resources. *IEEE Trans. Parallel Distrib. Syst.* **29**(6), 1401–1415. https://doi.org/10.1109/TPDS.2018.2793254 (2018).
24. Xu, H., Yu, W., Griffith, D. & Golmie, N. A survey on industrial Internet of Things: A cyber-physical systems perspective. *IEEE Access* https://doi.org/10.1109/access.2018.2884906 (2018).
25. Wu, G., Bao, W., Zhu, X. & Zhang, X. A general cross-layer cloud scheduling framework for multiple IoT computer tasks. *Sensors* https://doi.org/10.3390/s18061671 (2018).
26. Chen, L., Li, X. & Ruiz, R. Idle block-based methods for cloud workflow scheduling with preemptive and non-preemptive tasks. *Futur. Gener. Comput. Syst.* **89**, 659–669. https://doi.org/10.1016/j.future.2018.07.037 (2018).
27. Davis, R. & Burns, A. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.* **43**(4), 1–44. https://doi.org/10.1145/1978802.1978814 (2011).
28. Amalarethinam, D. I. G. & Beena, T. L. A. Customer facilitated cost-based scheduling (CFCSC) in cloud. *Procedia Comput. Sci.* **46**, 660–667. https://doi.org/10.1016/j.procs.2015.02.119 (2015).
29. Kao, M.-T., Cheng, Y.-H. & Kao, S.-J. An automatic decision-making mechanism for virtual machine live migration in private clouds. *Math. Probl. Eng.* **2014**, 328536. https://doi.org/10.1155/2014/328536 (2014).
30. Almuqren, L. et al. Hybrid metaheuristics with machine learning based botnet detection in cloud assisted internet of things environment. *IEEE Access* **11**, 115668–115676. https://doi.org/10.1109/ACCESS.2023.3322369 (2023).
31. Madni, S. H. et al. Hybrid gradient descent cuckoo search (HGDCS) algorithm for resource scheduling in IaaS cloud computing environment. *Clust. Comput.* **22**(1), 301–334. https://doi.org/10.1007/s10586-018-2856-x (2019).
32. Bolufé-Röhler, A. & Tamayo-Vera, D. Machine learning based metaheuristic hybrids for S-box optimization. *J. Ambient. Intell. Humaniz. Comput.* **11**, 5139–5152. https://doi.org/10.1007/s12652-020-01829-y (2020).
33. Jagadish Kumar, N. & Balasubramanian, C. Hybrid gradient descent golden eagle optimization (HGDGEO) algorithm-based efficient heterogeneous resource scheduling for big data processing on clouds. *Wirel. Pers. Commun.* **129**, 1175–1195. https://doi.org/10.1007/s11277-023-10182-0 (2023).
34. Seethalakshmi, V., Govindasamy, V. & Akila, V. Hybrid gradient descent spider monkey optimization (HGDSMO) algorithm for efficient resource scheduling for big data processing in heterogeneous environment. *J. Big Data* **7**, 49. https://doi.org/10.1186/s40537-020-00321-w (2020).
35. Fu, X. et al. Task scheduling of cloud computing based on hybrid particle swarm algorithm and genetic algorithm. *Clust. Comput.* **26**, 2479–2488. https://doi.org/10.1007/s10586-020-03221-z (2023).
36. Chhabra, A. et al. Optimizing bag-of-tasks scheduling on cloud data centers using hybrid swarm-intelligence meta-heuristic. *J. Supercomput.* **78**, 9121–9183. https://doi.org/10.1007/s11227-021-04199-0 (2022).
37. Zakarya, M., Gillam, L., Ali Khan, A., Rana, O. & Buyya, R. ApMove: A service migration technique for connected and autonomous vehicles. *IEEE Internet Things J.* https://doi.org/10.1109/JIOT.2024.3403415 (2024).
38. Zakarya, M. et al. epcAware: A game-based, energy, performance and cost-efficient resource management technique for multi-access edge computing. *IEEE Trans. Serv. Comput.* https://doi.org/10.1109/TSC.2020.3005347 (2022).
39. Khan, A. A. et al. A migration aware scheduling technique for real-time aperiodic tasks over multiprocessor systems. *IEEE Access* **7**, 27859–27873. https://doi.org/10.1109/ACCESS.2019.2901411 (2019).
40. Ayaz, A., Zakarya, M. & Khan, R. Energy-aware dynamic resource management in elastic cloud datacenters. *Simul. Model. Pract. Theory* https://doi.org/10.1016/j.simpat.2018.12.001 (2018).
41. Zakarya, M. et al. PerficientCloudSim: A tool to simulate large-scale computation in heterogeneous clouds. *J. Supercomput.* **77**, 3959–4013. https://doi.org/10.1007/s11227-020-03425-5 (2021).
42. Hussain, H. et al. Energy efficient real-time tasks scheduling on high-performance edge-computing systems using genetic algorithm. *IEEE Access* **12**, 54879–54892. https://doi.org/10.1109/ACCESS.2024.3388837 (2024).
43. Yang, C., Ding, Y., Hu, Z. & Ren, Z. Geometrized task scheduling and adaptive resource allocation for large-scale edge computing in smart cities. *IEEE Internet Things J.* https://doi.org/10.1109/jiot.2024.3525020 (2025).
44. Araújo, G. A., da Costa Bezerra, S. F. & da Rocha, A. R. Resource allocation based on task priority and resource consumption in edge computing. *J. Internet Serv. Appl.* https://doi.org/10.5753/jisa.2024.4026 (2024).

## Acknowledgements

The authors extend their appreciation to the Deanship of Research and Graduate Studies at King Khalid University for funding this work through the Large Research Project under grant number RGP2/109/46.

## Author contributions

Awad Bin Naeem and Biswaranjan Senapati: Data curation, methodology, formal analysis, software, investigation, validation, visualization, and writing-original draft. Jawad Rasheed and Onur Osman: Conceptualization, visualization, resources, writing-original draft, writing-review and editing. Jamel Baili: Conceptualization, visualization, writing-review and editing, and funding. All authors have read and agreed to the published version of the manuscript.

## Declarations

## Competing interests

The authors declare no competing interests.

## Additional information

**Correspondence** and requests for materials should be addressed to J.R.

**Reprints and permissions information** is available at www.nature.com/reprints.

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.