RATSS: A RISK-AWARE MULTI-OBJECTIVE TASK SCHEDULING STRATEGY FOR OPTIMIZED PERFORMANCE IN CLOUD-EDGE ENVIRONMENTS

Nisha Saini[1*], Jitender Kumar[2]

[1,2]Department of Computer Science and Engineering, Deenbandhu Chhotu Ram University of Science and Technology, Murthal (Sonipat), 131039, India.

Emails: 21001901005nisha@dcrustm.org[1], jitenderkumar.cse@dcrustm.org[2]

ABSTRACT

Efficient task scheduling in cloud-edge environments remains a significant challenge due to resource heterogeneity, varying workloads, and conflicting objectives of Cloud Service Providers (CSPs) and Cloud Service Users (CSUs). Existing multi-objective approaches primarily optimize average-based metrics such as makespan and waiting time while neglecting performance variability, leading to workload imbalance and unstable execution. To address this limitation, this study proposes a Risk-Aware Task Scheduling Strategy (RATSS) that incorporates variability into the optimization process by minimizing the standard deviation of makespan and waiting time, thereby reducing execution uncertainty. Reducing makespan variability improves workload consistency and resource utilization for CSPs, while minimizing waiting-time variability ensures fairness and reduced delays for CSUs. A composite metric, Satisfaction Ratio (SR), is introduced to jointly capture stability and fairness. The proposed approach is evaluated using the NASA iPSC workload dataset and compared with Non-dominated Sorting Genetic Algorithm (NSGA)-I, NSGA-II, Multi-Objective Particle Swarm Optimization, and the Strength Pareto Evolutionary Algorithm II. Results show average reductions of 48.05% and 63.23% in makespan and waiting-time variability, respectively, demonstrating improved stability, fairness, and overall system performance.

Keywords: Risk-Adjusted Return; Task Scheduling; System Performance; Resource Utilization; Quality of Service (QoS); Customer Satisfaction.

1.0 INTRODUCTION

The rapid evolution of cloud and edge computing paradigms has significantly reshaped distributed computing by enabling proximity-based service provisioning and reducing latency. Among these paradigms, edge computing has emerged as a critical extension of cloud infrastructure, facilitating computation offloading and real-time analytics closer to data sources. However, the decentralization of resources across heterogeneous and resource-constrained nodes introduces significant challenges in efficiently distributing workloads and provisioning resources [1]. Task scheduling in such environments remains a pivotal challenge due to dynamic workloads, limited computational capacity, and the need to balance execution latency with system throughput.

Task execution efficiency plays a vital role in overall system performance, particularly in minimizing makespan and improving resource allocation. Traditional scheduling approaches primarily focus on reducing the total execution time and often prioritize makespan as the primary optimization objective [2]. Although these approaches are effective for smaller workloads, they face significant challenges in complex, large-scale environments where heterogeneous resources and dynamic workloads require more adaptive and efficient scheduling mechanisms. In edge computing environments, heterogeneous Virtual Machine (VM) configurations and fluctuating workload intensities introduce performance variability [3], leading to resource overload, idleness, task starvation, and inconsistent execution behaviour.

To address these challenges, existing studies have explored heuristic, metaheuristic, and hybrid scheduling techniques. Heuristic approaches, such as list scheduling [4], critical path methods [5], and Round-Robin (RR) [6], provide computational efficiency and suitability for real-time environments. However, they often fail to effectively explore the solution space, leading to suboptimal scheduling decisions. Metaheuristic techniques, including Genetic Algorithms (GA) [7], Particle Swarm Optimization (PSO) [8], Ant Colony Optimization (ACO) [9], and the Grey Wolf Optimizer (GWO), improve solution quality through stochastic and population-based search mechanisms. However, these methods require careful parameter tuning and incur higher computational overhead, limiting their applicability in resource-constrained edge environments.

Hybrid scheduling approaches, which combine heuristic initialization with metaheuristic optimization, have gained attention for their ability to balance exploration and exploitation [8]-[11]. These approaches demonstrate

improvements in makespan reduction, resource usage, and workload balancing. However, most contemporary hybrid methods predominantly focus on optimizing average-based metrics, such as simple makespan, while neglecting the variability in task completion and waiting times. This limitation becomes crucial in large-scale environments, where performance variability leads to workload imbalance, prolonged waiting times, and inconsistent resource availability.

Prior studies, such as Mean Makespan Task Scheduling Strategy (MMTSS) [7], demonstrate that makespan variability increases substantially under extreme workloads and execution outliers. These conditions distort average performance metrics and amplify disparities across the VMs. As a result, scheduling efficiency deteriorates, leading to both underutilization and overutilization of resources.

Despite these advances, existing scheduling approaches remain inadequate in heterogeneous edge environments due to their reliance on average-based metrics and their inability to explicitly capture performance variability. This limitation results in workload imbalance, execution instability, and inefficient resource utilization. Therefore, the key problem addressed in this study is the lack of a variability-aware scheduling mechanism that ensures stable, efficient, and fair task execution under dynamic and resource-constrained conditions.

To address this problem, this study proposes the Risk-Aware Task Scheduling Strategy (RATSS), which extends beyond conventional average-based optimization by explicitly incorporating performance variability as a primary objective. Unlike traditional scheduling methods and MMTSS [7], RATSS integrates the standard deviation of makespan and waiting time as risk-aware performance indicators. These metrics enable the quantification of execution uncertainty and support more informed scheduling decisions.

Specifically, minimizing the standard deviation of makespan ensures consistent task completion across VMs, thereby improving workload distribution and resource utilization from the perspective of the Cloud Service Provider (CSP). Simultaneously, minimizing the standard deviation of waiting time enhances fairness in task execution, thereby benefiting Cloud Service Users (CSUs) by reducing task starvation and excessive response delays.

Furthermore, RATSS introduces a composite metric, termed Satisfaction Ratio (SR), to jointly capture execution stability and fairness. By improving VM load distribution and reducing performance variability, the proposed approach enhances overall system reliability and efficiency. From the CSP perspective, it improves operational efficiency and VM utilization, whereas from the CSU perspective, it ensures reliable service delivery, reduced response delays, and fair task execution. Consequently, the proposed mechanism offers a robust and scalable solution for achieving stable, efficient, and fair task scheduling in heterogeneous edge computing environments.

2.0 RELATED WORKS

The rapid advancement of edge computing has intensified research into efficient task scheduling strategies to minimize makespan and improve resource utilization. However, the inherently constrained and heterogeneous nature of edge environments introduces significant challenges in resource provisioning. In particular, over-provisioning leads to resource wastage and increased operational costs for CSPs, whereas under-provisioning results in increased latency, Service-Level Agreement (SLA) violations, and degraded Quality of Service (QoS) for CSUs. Consequently, both stakeholders experience reduced overall benefits, highlighting the need for balanced and intelligent scheduling mechanisms that simultaneously optimize resource utilization and service performance. A comparative summary of existing scheduling approaches and the proposed RATSS is presented in Table 1, clearly illustrating these limitations and the advantages of the proposed method.

Contemporary studies have explored heuristic, metaheuristic, hybrid, and learning-based approaches to address these challenges [12]-[15]. Heuristic-based methods are widely adopted due to their low computational overhead and suitability for real-time scheduling. Saif et al. [16] introduced a two-stage task offloading framework that combines Enhanced Task Offloading (ETO) with a Multi-Objective Firefly Algorithm (MFA) to improve the execution of critical tasks across edge and cloud layers. However, its focus on prioritized tasks limits its general applicability. Similarly, Dankolo et al. [17] employed a modified Flower Pollination Algorithm (FPA) with enhanced exploration and convergence properties via a dynamic switch probability and chaotic initialization. However, the robustness of this approach under fluctuating workloads and volatile network conditions has not been sufficiently validated. To optimize multiple objectives, Han et al. [18] presented the Cost and Makespan Scheduling of Workflows in the Cloud (CMSWC) algorithm for workflow scheduling in cloud environments, effectively balancing cost and makespan. However, its reliance on static cost models impairs flexibility in dynamic edge settings. Banerjee et al. [19] proposed MTD-DHJS, a dynamic time-prediction-based scheduling technique, showing promising scalability and responsiveness. However, the requirement for extensive training data poses deployment challenges.

Recent advancements have explored deep learning-based approaches for scheduling optimization. Wang et al. [20] introduced a Deep Reinforcement Learning (DRL)-based framework that incorporates a Graph Attention Network (GAT), Multilayer Perceptrons (MLP), and k-dimensional tree-based k-Nearest Neighbors (kNN) to map continuous decisions to discrete allocations. Despite its innovative design, its scalability under large-scale workloads remains a concern. Zou et al. [21] proposed ReflexPilot, which uses Proximal Policy Optimization (PPO) to manage executor

reuse and reduce task startup latency in edge-cloud systems. However, its performance under highly dynamic conditions requires further validation.

Metaheuristic techniques have been extensively applied to address the NP-hard nature of scheduling problems. Malti et al. [22] developed a hybrid PSO-SA-DE framework that integrates PSO, Simulated Annealing (SA), and Differential Evolution (DE), significantly improving makespan and resource utilization. However, the evaluation of the method in static environments limits its applicability to edge scenarios. Chhabra et al. [23] employed hybrid swarm intelligence for bag-of-tasks scheduling, targeting both cost and execution efficiency. However, the real-time and latency-sensitive performance of the method remains unexplored. Alboaneen et al. [24] proposed a framework for joint task scheduling and VM placement, yielding notable energy efficiency gains. However, it lacks adaptability to heterogeneous and mobile edge devices.

Hybrid approaches have gained increasing attention due to their ability to combine the strengths of multiple techniques. Lilhore et al. [25] developed a hybrid DRL-IACO-IWWO model, which integrates deep reinforcement learning with improved ant colony optimization and water wave optimization for adaptive workload management. Despite its improved balance between exploration and exploitation, its cross-cloud interoperability under variable workload intensities remains unexamined. Further advancements include reliability-aware and placement-sensitive models. Qin et al. [26] presented RA-MOMA, a multi-objective memetic algorithm that optimizes makespan, cost, and reliability, demonstrating its effectiveness in multi-cloud workflow scheduling. However, its real-time applicability is not well established. Zarei et al. [27] addressed Edge Server Placement (ESP) using a Mixed-Integer Nonlinear Programming (MINLP) formulation and ACO. However, the scalability of this approach for ultra-dense IoT networks remains a challenge.

Energy-aware scheduling has also been widely investigated. Li et al. [28] proposed a hybrid GA-PSO approach tailored for sparse edge-server deployments in mobile edge computing. While it achieved a significant improvement in workload balance, it demonstrated only modest gains in reducing task offloading delays. Wang et al. [29] extended this approach with a hybrid PSO-GA framework, incorporating a rescheduling mechanism to improve energy efficiency and makespan in device-edge-cloud systems. However, scalability and adaptability in dynamic settings were not fully validated.

Advances in reinforcement learning-based schedulers have also emerged. Zhang and Heath [30] introduced RLbDS, a recurrent neural network-based scheduler that improved both latency and utilization in dynamic environments. However, the computational cost of maintaining neural architectures poses deployment barriers in edge devices. Wang et al. [31] leveraged chaotic opposition-based learning within PSO to mitigate premature convergence, enhancing search diversity in task allocation. Despite its success, the model's limited attention to workload variability hinders its practical deployment in real-time scheduling. Mikram et al. [32] proposed HEPGA, a combination of HEFT, PSO, and GA, which achieved improved scheduling and resource utilization. However, its performance under highly heterogeneous and scaling conditions requires further study. In the context of deep learning tasks, Zhou et al. [33] explored the use of Tensor Virtual Machine (TVM) with adaptive scheduling to optimize remote sensing workflows, improving efficiency through salient point planning and adaptive management. However, the applicability of the method across varied deep learning architectures remains underexplored.

Overall, existing scheduling approaches have demonstrated significant improvements in objectives such as makespan reduction, energy efficiency, cost optimization, latency minimization, and resource utilization. Heuristic methods generally offer low computational complexity and fast decision-making, whereas metaheuristic and hybrid approaches provide enhanced search capabilities for complex scheduling problems. Similarly, learning-based techniques improve adaptability through data-driven decision processes. However, the majority of these studies evaluate scheduling performance primarily using average-based metrics and focus on optimizing a limited subset of objectives. As a result, the effects of execution-time variability, workload imbalance, and scheduling stability remain largely unexplored. Furthermore, improvements achieved for one stakeholder are often obtained without explicitly considering their impact on the other stakeholder, thereby limiting the overall effectiveness of existing scheduling solutions in heterogeneous cloud-edge environments.

Despite these advancements, a fundamental limitation persists across existing approaches. Most studies primarily focus on optimizing average-based metrics, such as makespan, cost, or energy consumption, while ignoring performance variability and its impact on system stability. Moreover, existing scheduling strategies tend to emphasize either CSP-centric objectives, such as VM utilization and cost efficiency, or CSU-centric objectives, such as latency and QoS, without providing a unified framework that ensures mutual benefits for both stakeholders. Consequently, these approaches often struggle to effectively mitigate over-provisioning and under-provisioning of resources, resulting in workload imbalance, inconsistent task execution, SLA violations, and degraded QoS. These limitations highlight the need for a variability-aware, risk-sensitive, and stakeholder-balanced scheduling mechanism capable of achieving stable, efficient, and fair task execution in cloud-edge environments.

Table 1: Comparison of existing scheduling approaches with proposed RATSS

Approach Type	Representative Methods	Optimization Focus	Handles Variability	CSP-CSU Balance	Scalability	Key Limitations
Heuristic	RR [6], List Scheduling [4], ETO [16]	Makespan, latency	No	Limited	High	Suboptimal solutions, poor adaptability
Metaheuristic	GA, PSO, ACO, GWO [7]-[10]	Makespan, cost	No	Limited	Moderate	High computational cost, parameter tuning
Hybrid	GA-PSO, DRL-IACO-IWWO [25]	Makespan	Limited	Partial	Moderate	Focus on average metrics, ignores scalability
DRL-Based	GAT-DRL [20], PPO [21]	Latency, resource efficiency	Limited	Partial	Low	High training cost, scalability issues
Energy-Aware	PSO-GA [28], [29]	Energy, makespan	No	CSP-focused	Moderate	Neglects QoS and fairness
Reliability-Aware	RA-MOMA [26]	Cost, reliability	Limited	Partial	Moderate	Limited real-time adaptability
Proposed RATSS	Risk-Aware Task Scheduling	Makespan stability, waiting-time fairness, SR	Yes (Standard deviation-based)	Balanced (CSP CSU)	High	Addresses variability, fairness, and stability jointly

3.0 PROPOSED RATSS MECHANISM

The proposed RATSS is designed to address the limitations of conventional task scheduling approaches that primarily emphasize execution efficiency while neglecting the impact of performance variability on scheduling stability and service quality. To achieve balanced scheduling decisions, RATSS formulates task scheduling as a multi-objective optimization problem that simultaneously considers execution performance, variability-related risk, and stakeholder-oriented service outcomes. The proposed mechanism integrates makespan, waiting time, variation in makespan, variation in waiting time, and SR within a unified optimization framework to support balanced benefits for both CSPs and CSUs. The framework, mathematical formulation, objective functions, and algorithmic implementation of the proposed mechanism are presented in the following subsections.

3.1 Framework of RATSS

As shown in Fig. 1, the proposed framework begins with workload and resource initialization, where tasks and VMs are characterized according to their computational requirements and processing capabilities. Subsequently, a risk-aware scheduling model is formulated to capture both execution efficiency and variability-related scheduling risks. To prevent optimization bias toward either provider-centric or user-centric objectives, RATSS incorporates the SR, which jointly considers mean makespan, makespan variation, and waiting-time variation. Consequently, scheduling decisions are directed toward solutions that simultaneously improve execution efficiency and scheduling stability while maintaining balanced benefits for both CSPs and CSUs.

The formulated optimization problem is solved using multi-objective evolutionary algorithms, namely Non-dominated Sorting Genetic Algorithm I (NSGA-I), NSGA-II, Multi-Objective Particle Swarm Optimization (MOPSO), and Strength Pareto Evolutionary Algorithm II (SPEA-II). These algorithms generate candidate scheduling solutions and explore diverse trade-offs among the considered objectives. The generated solutions are evaluated according to their ability to minimize execution delays and performance variability while maximizing the SR. Subsequently, Pareto-optimal schedules are identified, and the most suitable scheduling solution is selected for task allocation. The selected schedule is then executed by assigning tasks to the corresponding VMs according to the optimized allocation strategy.

Finally, the effectiveness of RATSS is assessed using five performance metrics: makespan, waiting time, standard deviation of makespan, standard deviation of waiting time, and SR. By simultaneously reducing execution delays and scheduling variability while improving user satisfaction, RATSS achieves a balanced trade-off between scheduling efficiency and execution stability. Consequently, the proposed mechanism improves service performance for CSUs while maintaining efficient scheduling outcomes for CSPs, thereby providing a risk-aware and stability-oriented scheduling solution for heterogeneous cloud-edge computing environments.

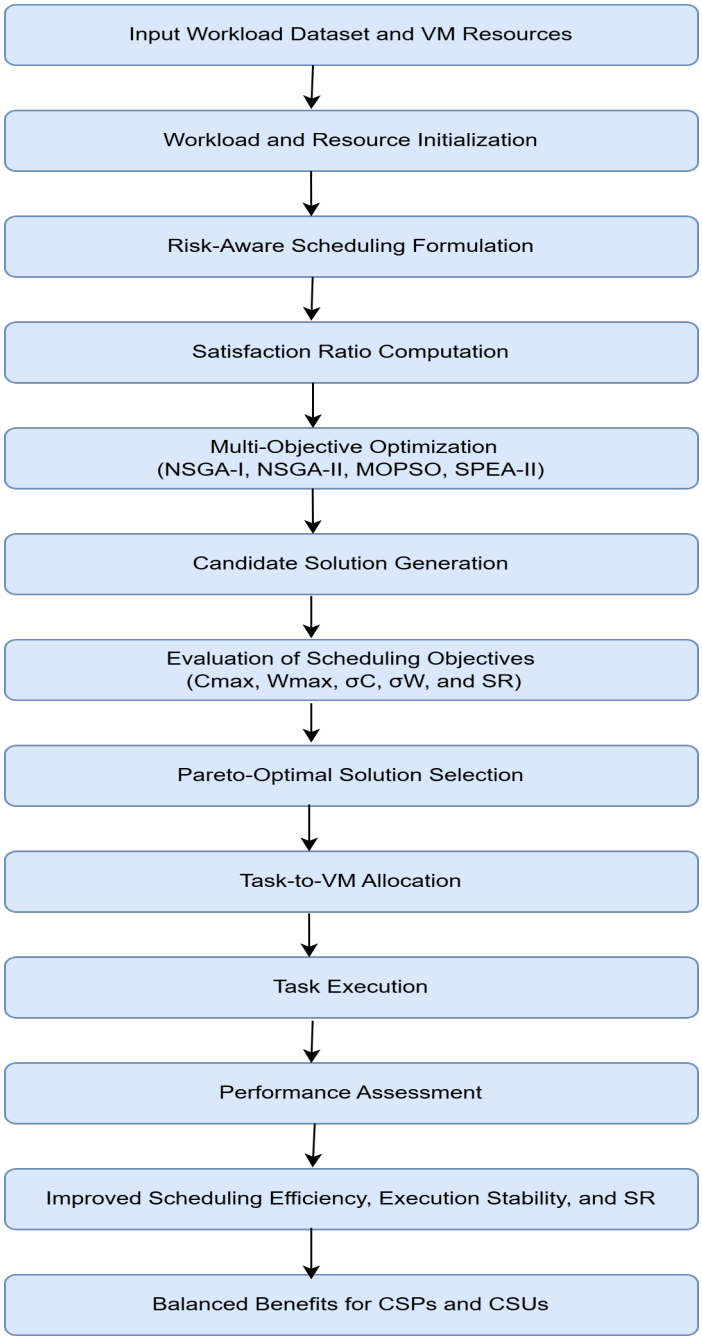

Fig. 1: Framework of the proposed RATSS

3.2 Mathematical Model and Objective Functions

The performance evaluation of edge services can be effectively validated through mathematical modelling, particularly by analyzing the makespan, waiting time, and associated variabilities. This section presents the formulation of mathematical models to compute the variabilities in both makespan and waiting time for a given set of

tasks deployed across multiple VMs. The system under consideration comprises a set of n tasks $T=\{T_1,T_2,T_3,.....,T_n\}$ and a set of m virtual machines $VM=\{VM_1,VM_2,VM_3,....,VM_m\}$. Let $P_{ij}\geq 0$ denotes the processing time of the task T_i on machine VM_j. $x_{ij}\in\{0,1\}$ is a binary decision variable such that $x_{ij}=1$ if task T_i is assigned to VM_j, and $x_{ij}=0$ otherwise. C_j represents the completion time (total load) of the machine VM_j, and $C_{\max}$ denotes the system makespan. $\overline{C}$ indicates the mean VM completion time. W_i be the waiting time experienced by the task T_i, and $\overline{W}$ represents the mean waiting time of tasks. Let σ_C denote the standard deviation of VM completion time, and σ_W represent the standard deviation of the waiting time of tasks. Let SR denote the Satisfaction Ratio. The multiple objectives are formally summarized as follows:

$$\min\left(C_{\max}\right) \tag{1}$$

$$\min\left(\sigma_C\right) \tag{2}$$

$$\min\left(\overline{W}\right) \tag{3}$$

$$\min\left(\sigma_W\right) \tag{4}$$

$$\max\left(SR\right) \tag{5}$$

The joint benefits of CSPs and CSUs can be captured by maximizing the satisfaction of both CSPs and CSUs, i.e.,

$$SR=\max\left(\frac{\overline{C}-\sigma_C}{\sigma_W}\right) \tag{6}$$

Subject to:

$$C_j=\sum_{i=1}^{n}x_{ij}P_{ij},\ \ \forall j\in\{1,2,.....,m\} \tag{7}$$

$$W_i=\sum_{j=1}^{m}x_{ij}\left(C_j-P_{ij}\right),\ \ \forall i\in\{1,2,....,n\} \tag{8}$$

$$\sum_{j=1}^{m}x_{ij}=1,\ \ \forall i \tag{9}$$

$$C_j\geq 0,\ W_i\geq 0,\ x_{ij}\in\{0,1\} \tag{10}$$

Constraint (7) computes the total execution time of VM_j. Constraint (8) computes the waiting time of T_i based on its execution order on the assigned VM. Constraint (9) ensures that each task is assigned to exactly one VM. Constraint (10) enforces non-negativity of completion and waiting times and defines the binary nature of the task-VM assignment variable.

For CSPs, the SR is enhanced when the difference between the $\overline{C}$ and the σ_C in the numerator is maximized. This difference will be maximum when σ_C (i.e., variability in makespan) is low. This difference increases as makespan variability decreases, promoting balanced distribution of the task load across all VMs and improving overall VM utilization. For CSUs, SR improvement is achieved by reducing the denominator σ_W, which represents the variability in waiting time. Lower waiting-time variability facilitates timely task execution and a more uniform distribution of waiting overheads, thereby leading to higher customer satisfaction. Overall, a higher SR value indicates an effective balance between the mean completion time and the risk of execution and waiting time. This formulation is conceptually aligned with the financial metric known as the Risk-Adjusted Return (RAR), which evaluates investment performance relative to its inherent risk [34]. The RAR is given as:

$$RAR=\frac{\text{Return On Investment}-\text{Risk Free Rate}}{\text{Risk}} \tag{11}$$

In financial analysis, the RAR captures the trade-off between expected return and variability. A similar interpretation is adopted in the proposed scheduling framework, where each component of the *SR* function corresponds to an analogous financial term:

- Return on Investment: Mean makespan $\left(\overline{C}\right)$
- Risk-Free Rate: Standard deviation of mean makespan $\left(\sigma_C\right)$

- Risk: Standard deviation of waiting time (σ_W)

This comparison allowed us to concurrently evaluate both performance (mean makespan) and risk (variability in makespan and waiting time). By defining our fitness function as an SR, we emphasize the dual benefits of reducing the makespan to enhance system performance for CSPs while simultaneously decreasing waiting times to mitigate risks and improve the QoS for CSUs.

3.3 Relationship Between Mean Makespan, Standard Deviation of Makespan, and Standard Deviation of Waiting Time

The reduction in $\overline{C}$ facilitates faster task execution and reduces the overall workload duration. However, minimizing the mean alone may lead to imbalances in VM utilization, as some VMs may complete tasks much earlier than others, resulting in load skew from outliers. To address this, RATSS minimizes the standard deviation of the makespan (σ_C), which quantifies the variability in the task completion times across VMs. A lower standard deviation implies a more balanced, uniform workload distribution, thereby reducing resource idling and the formation of bottlenecks in the system. In addition to execution efficiency, the RATSS targets waiting time optimization. The average waiting time $(\overline{W})$ represents the delay experienced by tasks before execution. Excessive wait times can degrade the system's responsiveness and increase user dissatisfaction. However, as with the makespan, minimizing $\overline{W}$ alone may not ensure fairness in the scheduling. Hence, RATSS incorporates the standard deviation of the waiting time (σ_W) as a separate objective. Reducing this variability minimizes the risk of starvation and ensures timely task responses, thereby ensuring an even distribution of waiting overheads across all CSUs.

3.3.1 The Mean Value of The Makespan Is Always Less Than or Equal to The Makespan (i.e., $\overline{C} \le C_{\max}$)

The completion time (C_j) of the VM_j is defined as

$$C_j = \sum_{i=1}^{n} P_{ij} x_{ij} \,, \tag{12}$$

The mean completion time for all VMs is denoted by $\overline{C}$, which is computed as

$$\overline{C} = \frac{1}{m} \sum_{j=1}^{m} C_j \,. \tag{13}$$

The overall makespan is the maximum VM completion time:

$$C_{\max} = \max\left\{ C_j \,\middle|\, 1 \le j \le m \right\}. \tag{14}$$

Consider a system with n tasks and m VMs, the makespan is therefore,

$$C_{\max} = \max\left(C_1, C_2, ..., C_m\right). \tag{15}$$

Since the mean is always less than or equal to the maximum value in any set of non-negative numbers [35], the following inequality holds:

$$\overline{C} \le C_{\max} \,, \tag{16}$$

i.e., the arithmetic mean of the machine completion times never exceeds their maximum. To formally establish that the mean completion time never exceeds the makespan, mathematical induction is applied.

Case 1: Single task on a single machine.
If only one machine is present, then $\{C_1\}$ is the singleton set of completion times, and $\overline{C} = C_1 = C_{\max}$. Thus, Eq. (16) holds with equality.

Case 2: k tasks executed on k machines.
We assume Eq. (16) holds for some $m = k$ machines, i.e., for any k non-negative numbers $C_1, C_2, ..., C_k$, it holds that $\frac{1}{k} \sum_{j=1}^{k} C_j \le \max_{1 \le j \le k} C_j$.

Case 3: $k+1$ tasks distributed across m machines.

We now consider $m = k+1$ machines with completion times $C_1, ..., C_k, C_{k+1}$. Therefore, the mean is denoted by

$$\overline{C}^{k+1} = \frac{1}{k+1}\sum_{j=1}^{k+1} C_j \ .$$

Let $C_{\max} = \max\left(C_1, ..., C_{k+1}\right)$. We observe that each $C_j \le C_{\max}$. By summing these inequalities over $j = 1, ..., k+1$ yields $\sum_{j=1}^{k+1} C_j \le (k+1) C_{\max}$. Dividing $k+1$ on both sides gives $\overline{C}^{k+1} \le C_{\max}$, which is precisely Eq. (16) for $m = k+1$. Thus, Eq. (16) holds for all positive integers m by mathematical induction. Let the deviation of each machine's completion time from the mean be

$$d_j = C_j - \overline{C}, \quad j = 1, ..., m \ , \tag{17}$$

Since the mean represents the equilibrium point of the distribution, the sum of deviations around the mean is zero.

$$\therefore \quad \sum_{j=1}^{m} d_j = 0 \tag{18}$$

The maximum deviation corresponds to the difference between the makespan and the mean:

$$\max\left(d_j\right) = C_{\max} - \overline{C} \tag{19}$$

As the mean is a central value in the distribution, the maximum deviation cannot be negative. Therefore,

$$C_{\max} - \overline{C} \ge 0 \tag{20}$$

This reaffirms the inequality:

$$\overline{C} \le C_{\max}$$

This relationship is crucial for scheduling, as the difference $\left(C_{\max} - \overline{C}\right)$ directly quantifies the extent of workload imbalance. Minimizing this difference reduces deviation among VMs, thereby equalizing their completion times. In practical terms, the closer the makespan moves toward the mean makespan, the more evenly tasks are distributed across VMs. This ensures efficient VM utilization and balanced task allocation, thereby preventing overloaded or idle VMs and enhancing system performance in edge computing environments.

3.3.2 Minimization of The Standard Deviation of The Finish Time of Machines Is Equivalent to The Minimum of The Makespan

The Standard deviation of the makespan $\left(\sigma_C\right)$ is represented as

$$\sigma_C = \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(C_j - \overline{C}\right)^2} \tag{21}$$

From Eq. (16), $\overline{C} \le C_{\max} \Rightarrow \overline{C}$ is the minimum makespan. By the standard deviation property, i.e., the Standard deviation is minimum when taken around the mean. Minimization of standard deviation is represented as:

$$\min\left(\sigma_C\right) = \min\sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(C_j - \overline{C}\right)^2} \tag{22}$$

The sum of squared deviations $\sum_{j=1}^{m}\left(C_j - \overline{C}\right)^2$ is always non-negative. It reaches its minimum (0) if and only if all $C_j = \overline{C}, \ \forall j$. If σ_C is minimized $\left(\sigma_C = 0\right)$, then

$$C_1 = C_2 = ...C_m = \overline{C} \tag{23}$$

Therefore, the makespan is

$$C_{\max} = \overline{C} \tag{24}$$

Consequently, $$\min\left(\sigma_C\right) \Rightarrow C_{\max} = \overline{C} \tag{25}$$

Minimizing the standard deviation brings the makespan closer to the mean, and since $\overline{C}$ is the minimum of the makespan (Eq. (16)). Overall, minimizing the standard deviation of machine completion times, i.e., $\sigma_C = \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(C_j - \overline{C}\right)^2}$ reduces the difference between the maximum completion time and the mean, i.e., minimizes

the makespan. Consequently, when $\sigma_C = 0$, the difference between the minimum makespan $\left(\overline{C}\right)$ and σ_C will be maximum. It ensures higher satisfaction of CSPs.

3.3.3 Minimization of The Standard Deviation of The Wait Time Ensures Minimization of The Simple Wait Time and The Mean Wait Time

We demonstrated that minimizing the standard deviation of the makespan minimizes both the simple and the mean makespan; the same holds for the waiting time. The standard deviation of the wait time is given as

$$\sigma_W = \sqrt{\frac{1}{k}\sum_{i=1}^{k}\left(W_i - \overline{W}\right)^2} \tag{26}$$

The reduction in the standard deviation of waiting time $\left(\sigma_W\right)$ guarantees that waiting times $\left(W_i\right)$ are closely grouped around their mean $\left(\overline{W}\right)$. This mitigates outliers, thereby minimizing both the maximum $\left(W_{\max}\right)$ and average wait times $\left(\overline{W}\right)$. Therefore,

$$\min\left(\sigma_W\right) \Rightarrow \min\left(W_{\max}\right) \wedge \min\left(\overline{W}\right) \tag{27}$$

3.4 RATSS Algorithm

Algorithm 1 presents the implementation procedure of RATSS. The algorithm iteratively evaluates candidate scheduling solutions using the objective functions defined in Eqs. (1)–(5) and employs a multi-objective optimization process to identify Pareto-optimal schedules. The final scheduling decision is selected from the Pareto-optimal solution set and used for task allocation. The computational complexity of RATSS is primarily determined by the employed multi-objective optimization algorithm. For a population size *N* and *I* optimization iterations, the overall computational complexity is approximately $O\left(I \times N^2\right)$, primarily due to the non-dominated sorting operations executed during each iteration.

Algorithm 1: Proposed RATSS Strategy.

Input:

$T = \{T_1, T_2, T_3, ..., T_n\}$ // Set of tasks

$VM = \{VM_1, VM_2, VM_3, ..., VM_m\}$ // Set of virtual machines

Output:

S* // Optimal task-to-VM allocation

Begin

1: Initialize tasks T and virtual machines VM
2: Generate an initial population P of candidate schedules S
3: while stopping criterion is not satisfied do
4: for each schedule S ∈ P do
5: Compute $C_{\max}\left(S\right)$
6: Compute $W_{\max}\left(S\right)$
7: Compute $\sigma_C\left(S\right)$ and $\sigma_W\left(S\right)$
8: Compute mean makespan $\overline{C}\left(S\right)$
9: Compute $SR\left(S\right) = \left(\overline{C}\left(S\right) - \sigma_C\left(S\right)\right) / \sigma_W\left(S\right)$
10: Evaluate objective functions using Eqs. (1)-(5)
11: end for
12: Apply the selected multi-objective optimizer

(NSGA-I, NSGA-II, MOPSO, SPEA-II)
to generate offspring schedules
13: Update population P
14: end while
15: Obtain Pareto-optimal solution set P*
16: Select optimal schedule S* from P*
17: Allocate tasks to VMs according to S*
18: Return S*
End

4.0 RESEARCH METHODS

The performance improvements achieved by RATSS are attributed to its variability-aware optimization mechanism. By minimizing the standard deviation of makespan, the approach ensures balanced workload distribution across VMs, thereby reducing resource underutilization and overloading, which directly benefits CSPs through improved resource utilization and operational efficiency. Similarly, minimizing the standard deviation of waiting time improves fairness in task scheduling by preventing excessive delays and task starvation, thereby benefiting CSUs through more consistent and timely task execution. This dual optimization enables RATSS to simultaneously address the objectives of both stakeholders, resulting in stable and consistent execution behavior and improved overall system performance. Furthermore, the reduction in variability limits extreme execution outliers, leading to more predictable scheduling behavior and enhanced consistency across varying workloads.

4.1 Experimental Setup

The experimental evaluation is conducted using workloads derived from the NASA Ames iPSC/860 log dataset, which contains more than 18,240 real-world scientific tasks [36]. Four workload sizes comprising 500, 1000, 1500, and 2000 tasks are considered to investigate scheduling performance under varying workload intensities. The scheduling experiments are performed in a virtualized environment consisting of six virtual machines (VMs), each configured with 2 GB RAM, 40 GB storage, and a single CPU core. For each workload size, tasks are scheduled using both conventional multi-objective optimization algorithms (NSGA-I, NSGA-II, MOPSO, and SPEA-II) and their corresponding RATSS-based implementations under identical experimental conditions. The resulting schedules are evaluated using makespan, waiting time, standard deviation of makespan, standard deviation of waiting time, and SR. To facilitate a clear comparative analysis, the performance results obtained from individual conventional algorithms and RATSS-based implementations are aggregated and analyzed using average values. All simulations are implemented in Python using Google Colab to ensure consistency and reproducibility. Table 2 summarizes the configuration parameters used in the experimental setup.

Table 2: Configuration of experimental parameters

Parameters	**Configured Values**
Tasks size	500-2000 tasks
Input file size	20-25 MB
RAM (per VM)	2 GB
Disk capacity (per VM)	40 GB
Number of VMs	6
CPU cores (per VM)	1 core
Simulator environment	Google Colab (Python 3.x)
Dataset used	NASA Ames iPCS/860 log dataset
Dataset type	Standardized real-world parallel scientific workload
Data size	18,240 tasks
Dataset file size	~1 MB (compressed) and 4 MB (uncompressed)
Algorithms used	MOPSO, NSGA-I, NSGA-II, SPEA2
Experiment repeats	30 independent runs

4.2 Baseline Methods and Result Aggregation Strategy

Conventional task scheduling methods often scalarize multiple conflicting objectives, such as makespan and waiting time, into a single objective using a weighted aggregation model [37], [38]. However, this approach imposes fixed trade-offs among objectives and fails to explicitly incorporate critical system-level constraints, such as resource heterogeneity, task completion constraints, and SLA violations, which are essential in edge scheduling scenarios. To ensure a methodologically sound and unbiased evaluation, the performance of RATSS is benchmarked with well-established Evolutionary Multi-Objective Optimization (EMO) algorithms that inherently handle conflicting objectives without prior scalarization. The evaluation is conducted using key performance metrics, including makespan, waiting time, standard deviation of makespan, standard deviation of waiting time, and SR.

The inclusion of conventional EMO algorithms, such as NSGA-I, NSGA-II, MOPSO, and SPEA-2, mitigates algorithm-specific bias and enables a fair comparison with established multi-objective schedulers. To improve interpretability and reduce the impact of stochastic variability inherent in evolutionary algorithms, performance metrics are averaged across all baseline algorithms and RATSS implementations for each task size. This aggregation provides a concise yet statistically meaningful comparison, while preserving consistent performance trends across varying workloads.

4.3 Evaluation Methods

The performance of the proposed RATSS is evaluated through a comparative analysis against conventional multi-objective scheduling approaches under identical experimental conditions. To ensure a fair and unbiased comparison, all scheduling algorithms are executed using the same workload dataset, VM configuration, and optimization settings described in Section 4.1. The effectiveness of each scheduling strategy is assessed using five complementary performance metrics, namely makespan, waiting time, standard deviation of makespan, standard deviation of waiting time, and SR. Collectively, these metrics evaluate execution efficiency, scheduling stability, workload balance, and service quality from the perspectives of both CSPs and CSUs.

Makespan and waiting time are employed to quantify scheduling efficiency by measuring the overall task completion time and the average delay experienced before task execution, respectively. Since average-based metrics alone do not adequately capture execution consistency, the standard deviations of makespan and waiting time are additionally evaluated to quantify execution-time variability and scheduling stability. Lower variability indicates a more balanced distribution of tasks across VMs, thereby reducing workload imbalance, minimizing execution uncertainty, and improving the reliability of scheduling decisions. Furthermore, the satisfaction ratio, defined in Eq. (6), integrates execution efficiency with variability-aware performance measures to provide a balanced evaluation of scheduling quality while simultaneously considering the interests of CSPs and CSUs.

The comparative evaluation is conducted for workload sizes ranging from 500 to 2000 tasks. For each workload, the performance of the proposed RATSS is compared with the average performance obtained from the baseline multi-objective optimization algorithms. This evaluation methodology enables a comprehensive assessment of the proposed mechanism in terms of efficiency, stability, robustness, and stakeholder-oriented scheduling performance under increasing workload intensity.

5.0 RESULTS

5.1 Makespan Analysis

Table 3 and Fig. 2 present the comparative average makespan results obtained using conventional scheduling algorithms and the proposed RATSS-based scheduling mechanism for different task sizes. For 500 tasks, the average makespan achieved by conventional methods is 72,143.75 s, whereas RATSS reduces it to 53,967.22 s. Similarly, for 1000, 1500, and 2000 tasks, the average makespan decreases from 164,922.50 s to 125,908.29 s, from 227,617.50 s to 178,967.99 s, and from 292,425.00 s to 230,553.62 s, respectively. The consistent reduction in makespan across all workload sizes indicates that RATSS improves scheduling efficiency by simultaneously considering execution performance and variability during schedule generation. By incorporating variability-aware objectives into the optimization process, RATSS promotes a more balanced distribution of tasks among available VMs, thereby reducing workload concentration and execution bottlenecks. Consequently, resources are utilized more effectively, resulting in shorter overall task completion times compared with conventional scheduling approaches.

Table 3: Makespan comparison between conventional and RATSS-based scheduling methods

Tasks	Average Makespan (Conventional) (sec)	Average Makespan (RATSS) (sec)
500	72,143.75	53,967.22
1000	164,922.50	125,908.29
1500	227,617.50	178,967.99
2000	292,425	230,553.62

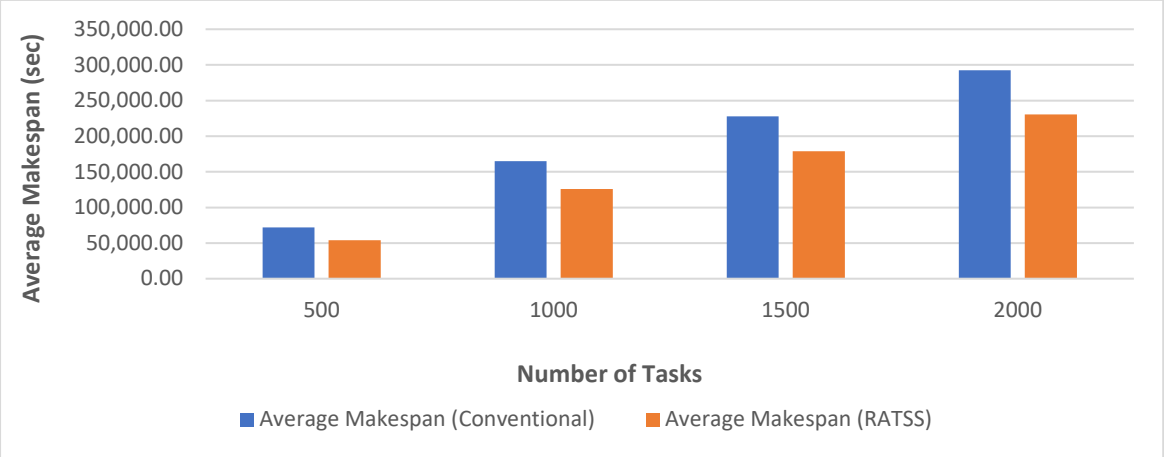

Fig. 2: Comparison of average makespan for conventional and RATSS-based scheduling approaches

5.2 Waiting Time Analysis

Table 4 and Fig. 3 present the comparative Average Waiting Time (AWT) results obtained using conventional scheduling algorithms and the proposed RATSS-based scheduling mechanism for different task sizes. For 500 tasks, the AWT decreases from 0.8361 s under conventional methods to 0.6358 s with RATSS. When the task size increases to 1000, RATSS reduces the AWT from 1.5286 s to 1.2709 s. A similar improvement is observed for 1500 tasks, where the AWT decreases from 2.1587 s to 1.8982 s. For a workload of 2000 tasks, RATSS achieves a reduction from 2.9034 s to 2.5411 s. The consistent reduction in waiting time across all workload sizes indicates that RATSS improves scheduling responsiveness by incorporating both execution efficiency and variability considerations into the optimization process. By promoting a more balanced allocation of tasks among available VMs, RATSS reduces queue accumulation and prevents excessive delays caused by uneven workload distribution. Consequently, tasks experience shorter waiting periods before execution, resulting in improved scheduling performance under increasing workload conditions.

Table 4: Average waiting time values for conventional and RATSS-based scheduling methods

Tasks	AWT (Conventional) (sec)	AWT (RATSS) (sec)
500	0.8361	0.6358
1000	1.5286	1.2709
1500	2.1587	1.8982
2000	2.9034	2.5411

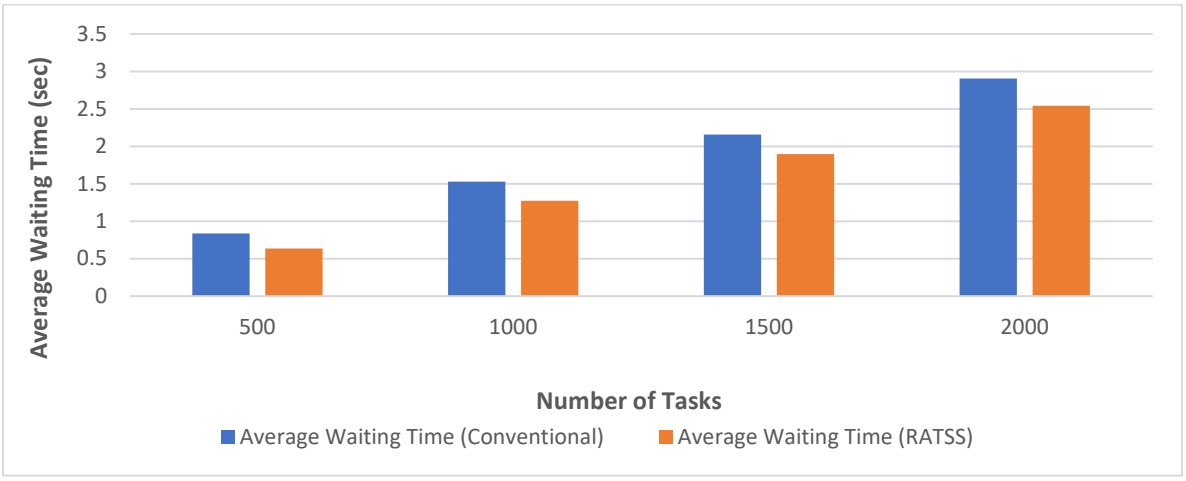

Fig. 3: Comparative analysis of average waiting time for conventional and RATSS-based scheduling approaches

5.3 Variation in Makespan Analysis

Table 5 and Fig. 4 report the average standard deviation of makespan for different task sizes. For 500 tasks, the standard deviation decreases from 57,144.21 s to 35,297.14 s, while for 1000 tasks it decreases from 95,808.70 s to 51,498.10 s. Similar reductions are observed for 1500 and 2000 tasks, where the variability declines from 133,334.61 s to 73,276.63 s and from 158,103.88 s to 82,092.45 s, respectively. The consistent reduction in makespan variability indicates that RATSS improves scheduling stability by explicitly incorporating variability-aware objectives into the optimization process. Unlike conventional approaches that primarily focus on execution efficiency, RATSS simultaneously considers both performance and risk-related factors during schedule generation, thereby promoting a more balanced distribution of workloads across available VMs. This balanced allocation reduces execution-time dispersion among resources and mitigates workload concentration, resulting in improved scheduling consistency and more effective resource utilization from the perspective of CSPs.

Table 5: Average standard deviation of makespan across evaluated scheduling approaches

Tasks	**Standard Deviation of Makespan (Conventional) (sec)**	**Standard Deviation of Makespan (RATSS) (sec)**
500	57,144.21	35,297.14
1000	95,808.70	51,498.10
1500	133,334.61	73,276.63
2000	158,103.88	82,092.45

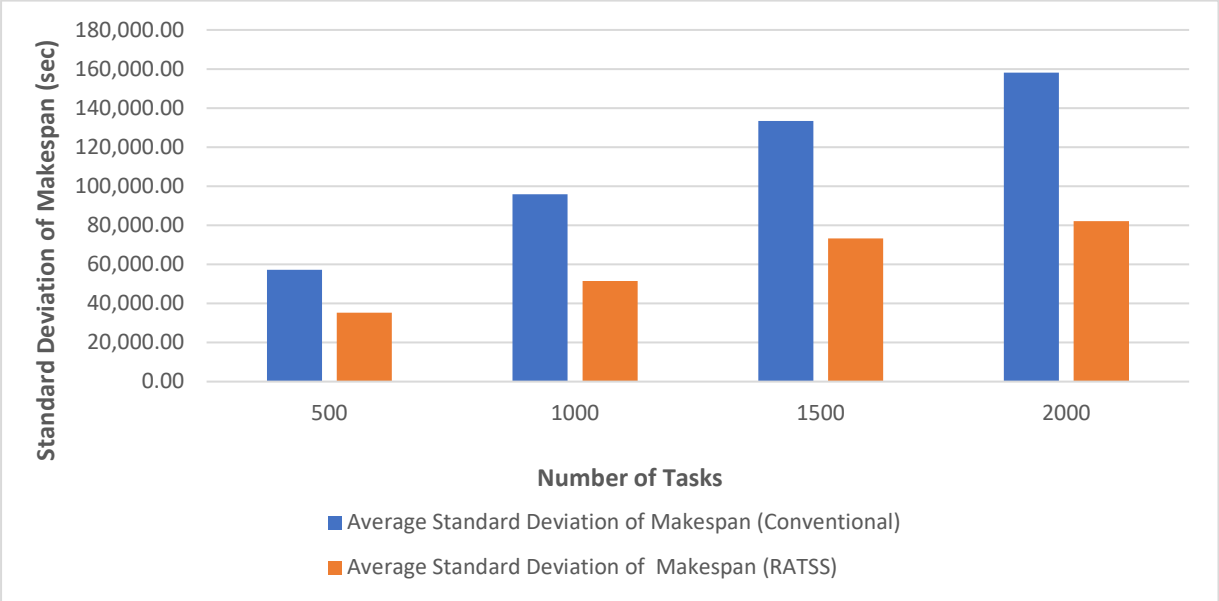

Fig. 4: Comparison of the standard deviation of makespan across scheduling algorithms

5.4 Variation in Waiting Time Analysis

Table 6 and Fig. 5 report the standard deviation of waiting time for conventional scheduling approaches and the proposed RATSS mechanism across different workload sizes. For 500 tasks, the standard deviation decreases from 0.3984 s to 0.1833 s. As the workload increases to 1000 and 1500 tasks, RATSS further reduces the variability from 0.9065 s to 0.3332 s and from 1.1694 s to 0.4491 s, respectively. For 2000 tasks, the standard deviation decreases from 1.3652 s under conventional methods to 0.5307 s with RATSS. The consistent reduction in waiting-time variability demonstrates the ability of RATSS to maintain stable scheduling behaviour under increasing workload conditions. By explicitly incorporating waiting-time variability into the optimization process, RATSS mitigates queue imbalance and prevents excessive waiting delays experienced by individual tasks. Consequently, task execution becomes more consistent across the workload, leading to improved scheduling fairness and a more reliable quality of service for CSUs.

Table 6: Average variability of waiting time across workloads

Tasks	Standard Deviation of Wait Time (Conventional) (sec)	Standard Deviation of Wait Time (RATSS) (sec)
500	0.3984	0.1833
1000	0.9065	0.3332
1500	1.1694	0.4491
2000	1.3652	0.5307

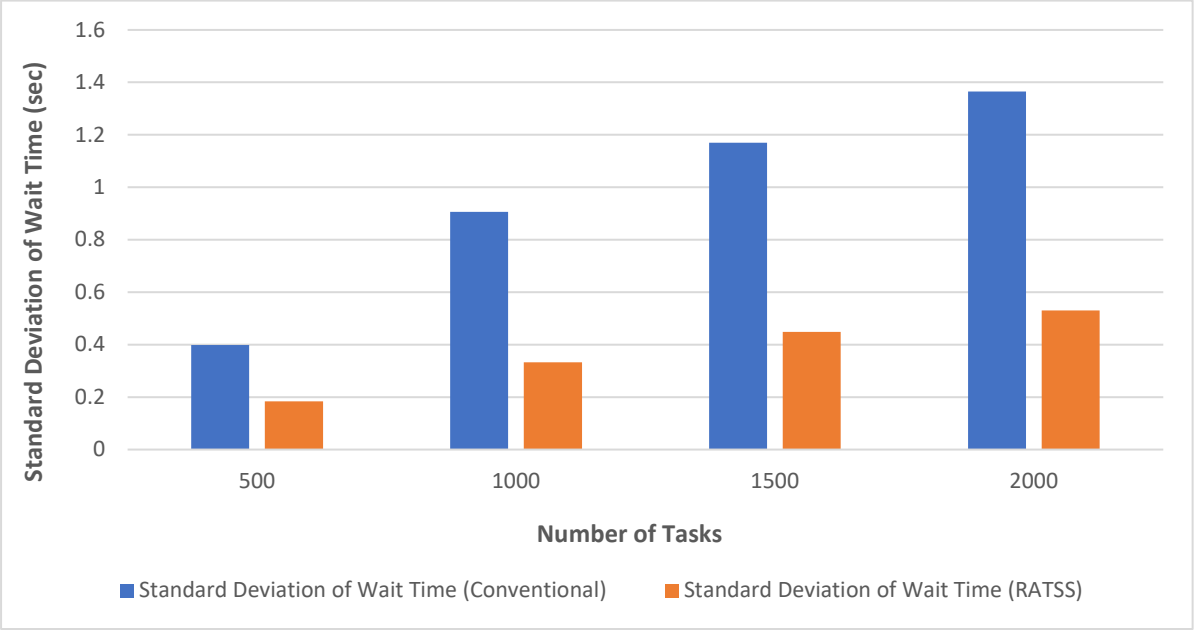

Fig. 5: Comparison of the standard deviation of waiting time across scheduling approaches

5.5 Satisfaction Ratio (SR) Analysis

Table 7 and Fig. 6 present the average SR results obtained using conventional scheduling approaches and the proposed RATSS mechanism across varying workload sizes. For 500 tasks, the SR increases from 30,575.27 to 40,127.81. Similar improvements are observed for 1000, 1500, and 2000 tasks, where the SR increases from 37,575.20 to 46,950.60, from 42,197.27 to 51,141.77, and from 46,555.25 to 56,480.21, respectively. The consistent improvement in SR demonstrates the effectiveness of RATSS in simultaneously enhancing scheduling efficiency and stability. Furthermore, the maximization of SR plays a crucial role in balancing the interests of CSPs and CSUs. Since SR is formulated using mean makespan, makespan variability, and waiting-time variability, higher SR values are achieved only when execution efficiency and scheduling stability improve simultaneously. The reduction in makespan variability benefits CSPs through improved VM utilization and balanced workload distribution, whereas the reduction in waiting-time variability benefits CSUs by providing more consistent service delivery and reducing the likelihood of excessive task delays. Consequently, the maximization of SR establishes a balanced trade-off between provider-oriented and user-oriented objectives, ensuring that performance improvements are not achieved at the expense of either stakeholder. These results confirm that RATSS promotes a more balanced and effective scheduling strategy across varying workload conditions.

Table 7: Mean satisfaction ratio across different workload sizes

Tasks	Average SR (Conventional)	Average SR (RATSS)
500	30,575.27	40,127.81
1000	37,575.20	46,950.60
1500	42,197.27	51,141.77
2000	46,555.25	56,480.21

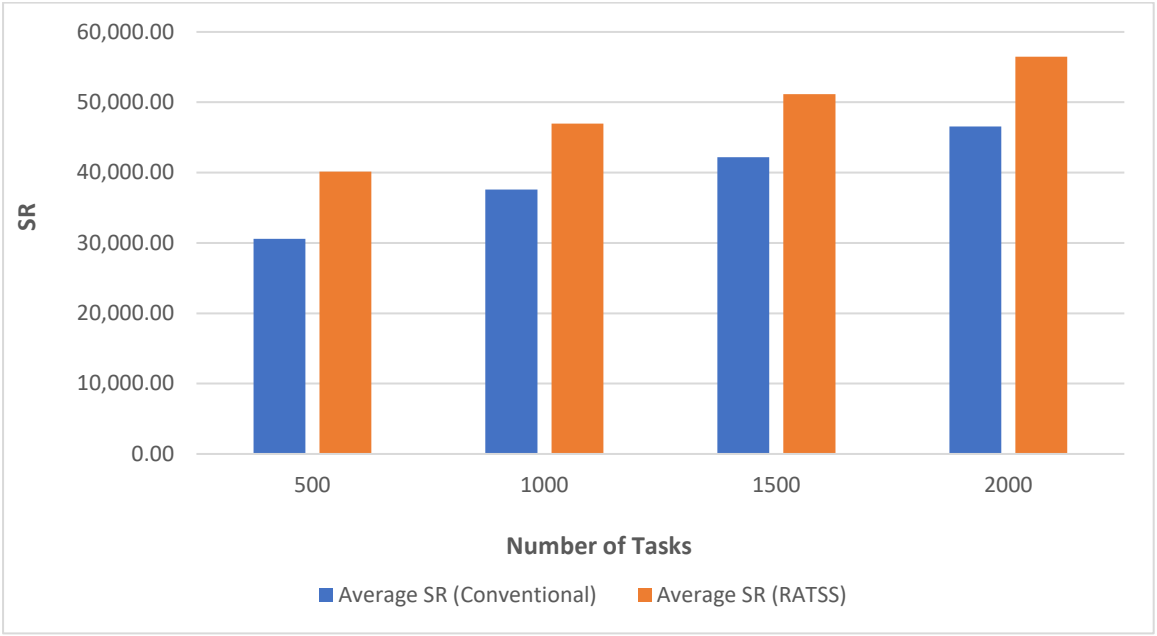

Fig. 6: Satisfaction Ratio (SR) comparison illustrating the balance between scheduling stability and fairness

5.6 Overall Performance Discussion

The effectiveness of the proposed RATSS mechanism is further examined through the combined reduction in makespan variability and waiting-time variability under increasing workload conditions. The comparative results obtained using conventional scheduling approaches and RATSS-based implementations are summarized in Table 8 and illustrated in Fig. 7. The percentage reduction in variability is computed using Eq. (28).

$$Variability\,\mathrm{Re}duction(\%) = \frac{Conventional\ Average - RATSS\ Average}{Conventional\ Average} \times 100 \tag{28}$$

As shown in Table 8, RATSS consistently achieves substantial reductions in makespan variability across all workload sizes. The standard deviation of makespan decreases by 38.24%, 46.26%, 45.05%, and 48.05% for workloads of 500, 1000, 1500, and 2000 tasks, respectively. These improvements indicate that RATSS effectively limits execution-time dispersion among VMs by incorporating makespan variability as an explicit optimization objective. Consequently, workload distribution becomes more balanced, reducing the possibility of VM overloading and underutilization. From the perspective of CSPs, lower makespan variability contributes to improved VM utilization, more stable resource allocation, and enhanced operational efficiency.

A similar trend is observed for waiting-time variability. RATSS reduces the standard deviation of waiting time by 54.00%, 63.23%, 61.59%, and 61.12% for workloads of 500, 1000, 1500, and 2000 tasks, respectively. Although the percentage reduction does not increase monotonically with workload size, this behaviour is expected because the reduction is computed relative to the corresponding conventional average, as defined in Eq. (28). As workload intensity increases, both conventional and RATSS-based waiting-time variability values increase in absolute terms. Consequently, a larger absolute reduction may correspond to a slightly lower percentage reduction when the baseline variability is also larger. Therefore, the minor fluctuations observed across workload sizes do not indicate performance degradation; rather, they reflect the relative nature of percentage-based comparison.

Table 8: Percentage reduction in makespan and waiting-time variability achieved by RATSS

Tasks	Makespan Variability Reduction (%)	Waiting Time Variability Reduction (%)
500	38.24	54
1000	46.26	63.23
1500	45.05	61.59
2000	48.05	61.12

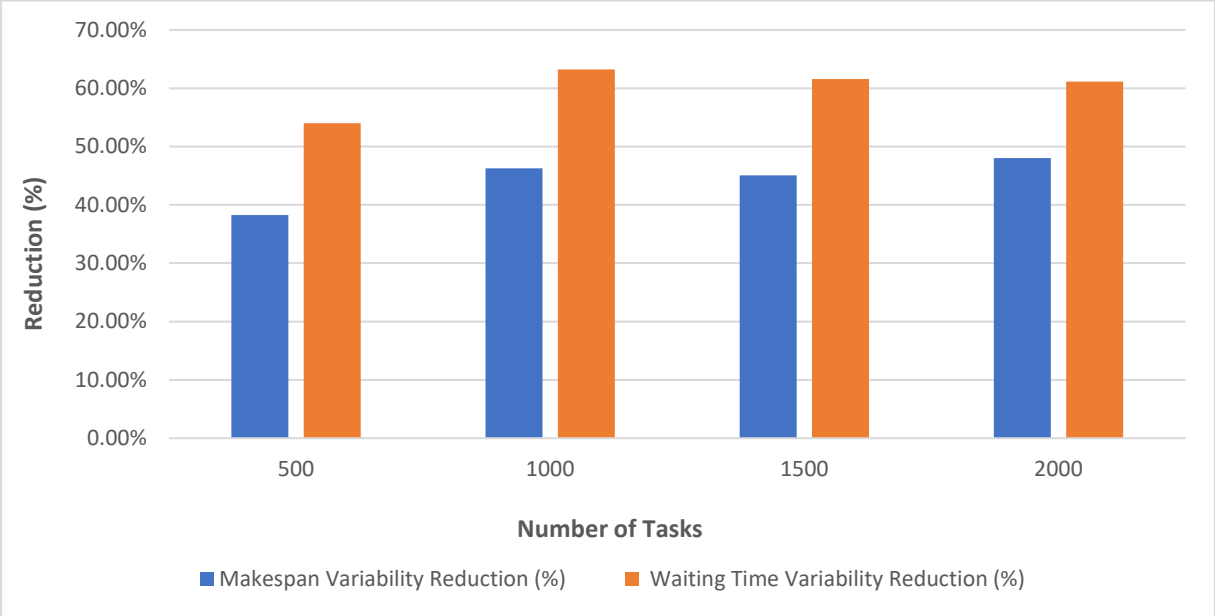

Fig. 7: Comparative improvement in scheduling variability achieved by RATSS

The performance improvements achieved by RATSS are attributed to its risk-aware multi-objective optimization strategy. Unlike conventional scheduling approaches that primarily focus on execution efficiency, RATSS simultaneously considers makespan, waiting time, makespan variability, waiting-time variability, and SR during schedule generation. The inclusion of variability-related objectives enables the identification of schedules that not only reduce execution delays but also maintain stable workload distribution across available VMs. Consequently, execution-time dispersion is reduced, leading to lower makespan variability and improved VM utilization from the perspective of CSPs. Similarly, the explicit minimization of waiting-time variability reduces queue imbalance and excessive waiting delays, thereby providing more consistent service delivery and improved QoS for CSUs. Overall, the results demonstrate that RATSS improves scheduling efficiency, stability, and robustness by jointly optimizing performance and variability objectives, thereby establishing a balanced trade-off between the interests of CSPs and CSUs.

6.0 CONCLUSION

This study addresses the challenges of task scheduling in heterogeneous edge environments, with a focus on improving both system efficiency and user satisfaction. A risk-aware task scheduling strategy is proposed that integrates a multi-objective optimization mechanism that balances execution efficiency, fairness, and stability. Unlike conventional approaches that rely on average-based metrics, RATSS integrates variability-aware indicators by minimizing the standard deviation of makespan and waiting time. Additionally, the SR is introduced to provide a balanced evaluation of performance from both CSP and CSU perspectives.

By reducing performance variability and improving workload distribution, the proposed approach effectively mitigates issues such as VM overloading, underutilization, and task starvation, thereby enhancing system reliability and consistency. Comparative evaluation against established algorithms, including NSGA-I, NSGA-II, MOPSO, and SPEA2, demonstrates that RATSS achieves average reductions of 48.05% in makespan variability and 63.23% in waiting time variability. These results validate the effectiveness of RATSS in improving scheduling stability, fairness, and overall system performance.

Although the proposed RATSS demonstrates significant improvements in variability-aware task scheduling, its applicability can be further enhanced. Future work will focus on extending the mechanism to support real-time and latency-sensitive applications in highly dynamic cloud-edge environments.

REFERENCES

[1] K. Malhotra, D. Gupta, S. Goel, and A. K. Tripathi, "Bi-objective flow shop scheduling with equipotential parallel machines," *Malaysian Journal of Mathematical Sciences*, vol. 16, no. 3, 2022, pp. 451–470, doi: 10.47836/mjms.16.3.04.

[2] A. A. Sharawy, R. H. Sakr, W. Eladrosy, and M. F. Alrahmawy, "WS-SSA: workflow scheduling in cloud computing using salp swarm algorithm," *Scientific Reports*, vol. 16, no. 1, 2026, p. 13402, doi: 10.1038/s41598-026-48037-w.

[3] T. Hagras and G. A. El-Sayed, "A fault-tolerant and load-balancing scheduler for independent tasks on cloud-based virtual machines," *Cluster Comput*, vol. 29, no. 1, Feb. 2026, p. 61, doi: 10.1007/s10586-025-05857-1.

[4] R. Madhura, V. R. Uthariaraj, and B. L. Elizabeth, "An efficient list-based task scheduling algorithm for heterogeneous distributed computing environment," *Softw Pract Exp*, vol. 53, no. 2, Feb. 2023, pp. 390–412, doi: 10.1002/spe.3153.

[5] B. Wu, Z. Xiao, P. Lin, Z. Tang, and K. Li, "Critical path awareness techniques for large-scale graph partitioning," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 3, 2023, pp. 412–422, doi: 10.1109/TSUSC.2023.3263172.

[6] C. Sharma, S. Sharma, S. Kautish, S. A. Alsallami, E. M. Khalil, and A. W. Mohamed, "A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time," *Alexandria Engineering Journal*, vol. 61, no. 12, 2022, pp. 10527–10538, doi: 10.1016/j.aej.2022.04.006.

[7] N. Saini and J. Kumar, "Mean makespan task scheduling approach for the edge computing environment," *International Journal of Electrical and Computer Engineering (IJECE),* vol. 14, no. 4, 2024, pp. 4714–4720, doi: 10.11591/ijece.v14i4.pp4714-4720.

[8] S. D. Vispute and P. Vashisht, "Energy-Efficient Task Scheduling in Fog Computing Based on Particle Swarm Optimization," *SN COMPUT. SCI.,* vol. 4, no. 4, May 2023, p. 391, doi: 10.1007/s42979-022-01639-3.

[9] Y.-T. Chuang and Y.-T. Hung, "A real-time and ACO-based offloading algorithm in edge computing," *Journal of Parallel and Distributed Computing,* vol. 179, 2023, p. 104703, doi: 10.1016/j.jpdc.2023.04.004.

[10] G. Agarwal, S. Gupta, R. Ahuja, and A. K. Rai, "Multiprocessor task scheduling using multi-objective hybrid genetic Algorithm in Fog–cloud computing," *Knowledge-Based Systems*, vol. 272, 2023, p. 110563, doi: 10.1016/j.knosys.2023.110563.

[11] E. Mirsadeghi and S. Khodayifar, "Hybridizing particle swarm optimization with simulated annealing and differential evolution," *Cluster Comput*, vol. 24, no. 2, June 2021, pp. 1135–1163, doi: 10.1007/s10586-020-03179-y.

[12] Z. Azmi, N. Haque, and S. Murad, "Comparative analysis of task scheduling in multi-tier fog-cloud computing: from classical approaches to greedy multi-objective optimization," *Journal of King Saud University Computer and Information Sciences,* 2026, doi: 10.1007/s44443-026-00649-y.

[13] N. Saini and J. Kumar, "Towards Efficient Resource Management in Edge Environments: Performance Analysis of Static, Dynamic, and Hybrid Approaches," *IAENG International Journal of Applied Mathematics*, vol. 56, no. 1, 2026, pp. 218-230.

[14] H. Altalhoni et al., "A novel artificial intelligence based dynamic task scheduling and load awareness," *Cluster Computing*, vol. 29, no. 3, 2026, p. 159, doi: 10.1007/s10586-026-05948-7.

[15] R. Salimi, S. Azizi, and M. Shojafar, "Dynamic and Resource-aware Task Scheduling in Fog-Cloud Environments via an Improved Priority-aware Genetic Algorithm," *Journal of Grid Computing*, vol. 24, no. 2, 2026, p. 5, doi: 10.1007/s10723-025-09821-6.

[16] F. A. Saif, R. Latip, Z. M. Hanapi, K. Shafinah, A. S. Kumar, and A. S. Bajaher, "Multi-Objectives firefly algorithm for task offloading in the Edge-Fog-Cloud computing," *IEEE Access*, vol. 12, 2024, pp.159561-159578, doi: 10.1109/ACCESS.2024.3488032.

[17] N. M. Dankolo, N. H. M. Radzi, N. H. Mustaffa, N. A. Osman, D. Gabi, and M. N. Yusuf, "Enhanced task scheduling and resource allocation in edge-cloud continuum using modified flower pollination algorithm," *IEEE Access*, vol. 12, 2024, pp.162299-162310, doi: 10.1109/ACCESS.2024.3490177.

[18] P. Han, C. Du, J. Chen, F. Ling, and X. Du, "Cost and makespan scheduling of workflows in clouds using list multiobjective optimization technique," *Journal of Systems Architecture*, vol. 112, 2021, p. 101837, doi: 10.1016/j.sysarc.2020.101837.

[19] P. Banerjee et al., "MTD-DHJS: makespan-optimized task scheduling algorithm for cloud computing with dynamic computational time prediction," *IEEE Access*, vol. 11, 2023, pp. 105578–105618, doi: 10.1109/ACCESS.2023.3318553.

[20] Z. Wang, W. Zhan, H. Duan, G. Min, and H. Huang, "Deep Reinforcement Learning-Based Continuous Workflows Scheduling in Heterogeneous Environments," *IEEE Internet of Things Journal*, vol. 12, no. 10, 2025, pp. 14036-14050, doi: 10.1109/JIOT.2024.3524506.

[21] W. Zou, Z. Zhang, N. Wang, Y. Tian, and L. Tian, "Reflexpilot: Startup-aware dependent task scheduling based on deep reinforcement learning for edge-cloud collaborative computing," *IEEE Transactions on Cloud Computing,* vol. 13, no. 2, April-June 2025, pp. 641-654, doi: 10.1109/TCC.2025.3555231.
[22] A. N. Malti, M. Hakem, and B. Benmammar, "A new hybrid multi-objective optimization algorithm for task scheduling in cloud systems," *Cluster Comput*, vol. 27, no. 3, June 2024, pp. 2525–2548, doi: 10.1007/s10586-023-04099-3.
[23] A. Chhabra, K.-C. Huang, N. Bacanin, and T. A. Rashid, "Optimizing bag-of-tasks scheduling on cloud data centers using hybrid swarm-intelligence meta-heuristic," *J Supercomput*, vol. 78, no. 7, May 2022, pp. 9121–9183, doi: 10.1007/s11227-021-04199-0.
[24] D. Alboaneen, H. Tianfield, Y. Zhang, and B. Pranggono, "A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers," *Future Generation Computer Systems,* vol. 115, 2021, pp. 201–212, doi: 10.1016/j.future.2020.08.036.
[25] U. K. Lilhore et al., "Hybrid DRL-Enhanced ACO-WWO for Efficient Resource Allocation and Load-Balancing in Cloud Computing," *Int J Comput Intell Syst*, vol. 18, no. 1, June 2025, p.148, doi: 10.1007/s44196-025-00882-9.
[26] S. Qin, D. Pi, Z. Shao, Y. Xu, and Y. Chen, "Reliability-aware multi-objective memetic algorithm for workflow scheduling problem in multi-cloud system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, 2023, pp. 1343–1361, doi: 10.1109/TPDS.2023.3245089.
[27] S. Zarei, S. Azizi, and A. Ahmed, "Optimizing edge server placement and load distribution in mobile edge computing using ACO and heuristic algorithms," *J Supercomput*, vol. 81, no. 1, Jan. 2025, p.257, doi: 10.1007/s11227-024-06780-9.
[28] W. Li, X. Sun, B. Wan, H. Liu, J. Fang, and Z. Wen, "A hybrid GA-PSO strategy for computing task offloading towards MES scenarios," *PeerJ Computer Science*, vol. 9, 2023, p. e1273, doi: 10.7717/peerj-cs.1273.
[29] Y. Wang, P. Zhang, B. Wang, Z. Zhang, Y. Xu, and B. Lv, "A hybrid PSO and GA algorithm with rescheduling for task offloading in device–edge–cloud collaborative computing," *Cluster Comput*, vol. 28, no. 2, Apr. 2025, p.101, doi: 10.1007/s10586-024-04851-3.
[30] Y. Zhang and R. W. Heath, "Reinforcement learning-based joint user scheduling and link configuration in millimeter-wave networks," *IEEE Transactions on Wireless Communications*, vol. 22, no. 5, pp. 3038–3054, 2022, doi: 10.1109/TWC.2022.3215922.
[31] L. Wang, D. Tian, X. Gou, and Z. Shi, "Hybrid particle swarm optimization with adaptive learning strategy," *Soft Comput*, vol. 28, no. 17–18, pp. 9759–9784, Sept. 2024, doi: 10.1007/s00500-024-09814-9.
[32] H. Mikram, S. El Kafhali, and Y. Saadi, "HEPGA: A new effective hybrid algorithm for scientific workflow scheduling in cloud computing environment," *Simulation Modelling Practice and Theory*, vol. 130, p. 102864, 2024, doi: 10.1016/j.simpat.2023.102864.
[33] Y. Zhou, L. Chen, Y. Liu, F. Ma, and F. Zhang, "Efficient Compilation Method for Remote Sensing Deep Learning Models Based on Search Optimization and Adaptive Clustering," *IEEE Transactions on Geoscience and Remote Sensing*, 2025, doi: 10.1109/TGRS.2025.3560604.
[34] P. Dupuy, "Risk-adjusted return managed carry trade," *Journal of Banking & Finance*, vol. 129, 2021, p. 106172, doi: 10.1016/j.jbankfin.2021.106172.
[35] S. C. Gupta and V. K. Kapoor, *Fundamentals of mathematical statistics*. Sultan Chand & Sons, 2020.
[36] Parallel Workloads Archive: NASA Ames iPSC/860. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/l_nasa_ipsc/
[37] S. Gupta and R. S. Singh, "User-defined weight-based multi-objective task scheduling in cloud using whale optimization algorithm," *Simulation Modelling Practice and Theory*, vol. 133, 2024, p. 102915, doi: 10.1016/j.simpat.2024.102915.
[38] A. Belgacem and K. Beghdad-Bey, "Multi-objective workflow scheduling in cloud computing: trade-off between makespan and cost," *Cluster Computing*, vol. 25, no. 1, 2022, pp. 579–595, doi: 10.1007/s10586-021-03432-y.
[39] N. Saini and J. Kumar, "Cost-Efficient and Latency-Aware Edge-Cloud Task Scheduling," *IEEE Internet Computing*, vol. 30, no. 1, Jan.-Feb. 2026, pp. 47–58, doi: 10.1109/MIC.2025.3648839.