SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

Adaptive Pre-emption-Aware Hierarchical Scheduling with Recovery Queues for Hadoop YARN

Bhavana P[1], Shashikumar D R[2]

[1]Cambridge Institute of Technology, Bengaluru, Visvesvaraya Technological University, Belagavi - 590018, India. https://orcid.org/0009-0007-8958-4653, E-mail: potli.bhavana13@gmail.com
[2]Sai Vidya Institute of Technology, Bengaluru, Visvesvaraya Technological University, Belagavi - 590018, India. https://orcid.org/0009-0007-6807-8161, E-mail: shashikumardr99@gmail.com

Abstract
Efficient job scheduling remains a fundamental challenge in Hadoop YARN because dynamic workloads, frequent resource contention, and task pre-emption often lead to increased waiting time, redundant execution, and reduced resource utilization. Conventional scheduling policies primarily focus on resource allocation and job ordering but provide limited support for preserving execution progress after task interruption. This paper presents an Adaptive Pre-emption-Aware Hierarchical Scheduling with Recovery Queues (APHRQ) framework to improve scheduling efficiency through adaptive decision-making and recovery-aware execution. The proposed framework combines adaptive score-based job selection, controlled pre-emption, checkpoint-assisted recovery, a dedicated Recovery Queue, and a Protected Queue to enable interrupted jobs to resume execution from their previously saved state while preventing repeated interruptions. A Recovery-Aware Fairness Index (RAFI) is also introduced to evaluate fairness by considering the scheduling opportunities afforded to interrupted workloads, in addition to conventional resource-allocation fairness. The proposed framework was evaluated using a Python-based Hadoop YARN simulation developed with Streamlit under normal, pre-emption-heavy, and critical workload scenarios. Comparative evaluation against FIFO, Fair Scheduler, Capacity Scheduler, Shortest Job First (SJF), Shortest Remaining Time First (SRTF), and a pre-emption-based scheduler without recovery support demonstrates that APHRQ improves throughput, resource utilization, recovery efficiency, and recovery-aware fairness while reducing redundant re-computation and starvation under dynamic workload conditions. The results indicate that integrating adaptive scheduling with checkpoint-assisted recovery provides a balanced and efficient scheduling strategy for Hadoop YARN environments.

Keywords: Adaptive Scheduling, Checkpoint Recovery, Hadoop YARN, Pre-emption, Recovery Queue.

1. Introduction

The rapid growth of data generated by IoT devices, social media platforms, enterprise applications, e-commerce services, and scientific computing environments has increased demand for efficient, large-scale data processing systems. Organizations today deal with massive volumes of structured and unstructured data that must be processed within limited time constraints while maintaining acceptable performance and resource utilization. Apache Hadoop has emerged as one of the most widely used frameworks for distributed data processing because of its scalability, fault tolerance, and ability to execute data-intensive applications across clusters of commodity hardware. To improve resource management and support multiple processing frameworks, Hadoop introduced Yet Another Resource Negotiator (YARN), which separates resource allocation from application execution, enabling efficient sharing of cluster resources.

Although YARN provides a flexible resource management framework, efficient scheduling remains a challenging problem. In a typical Hadoop cluster, multiple users submit jobs with different priorities, execution times, and resource requirements. Some jobs are short and interactive, while others are long-running and resource-intensive. As the workload increases, the scheduler must decide how to allocate available resources among competing jobs without affecting overall cluster performance. Poor scheduling decisions may lead to increased waiting time, low resource utilization, reduced throughput, and longer job completion times. Traditional YARN schedulers such as FIFO, Fair Scheduler, and Capacity Scheduler are widely used in practice; however, their performance often degrades when workload characteristics change dynamically or when cluster resources are heterogeneous in nature [1-2].

Another important challenge arises when high-priority jobs arrive while cluster resources are already occupied by running tasks. In such situations, pre-emption is commonly used to reallocate resources from lower-priority jobs to more urgent workloads. This approach improves responsiveness and helps maintain fairness among users. However, pre-emption also introduces additional overhead because interrupted tasks may lose part of their execution progress. Frequent interruptions can increase completion time, reduce scheduling efficiency, and generate unnecessary computation. Studies on pre-emptive scheduling have shown that controlling pre-emption overhead is essential for improving overall scheduling performance [3]. Therefore, simply introducing pre-emption is not sufficient; an effective mechanism is also required to manage interrupted tasks efficiently.

Hadoop already includes several fault-tolerance mechanisms that help recover from node failures and task failures during execution. These mechanisms mainly rely on replication and task re-execution to ensure reliability in distributed environments [4]. While such approaches are effective for handling failures, scheduler-induced interruptions represent a different scenario. A preempted task is not necessarily a failure; it is only paused to release resources for a higher-priority workload. Treating these interrupted tasks in the same way as failed tasks can lead to unnecessary re-execution and additional resource consumption. As workload diversity increases, this limitation becomes more significant and can negatively affect cluster performance.

Recent developments in Hadoop scheduling have focused primarily on improving resource allocation, energy efficiency, and adaptive workload management. Several approaches have demonstrated notable improvements in resource utilization and execution performance through adaptive scheduling strategies, while others have concentrated on reducing energy consumption and meeting application deadlines [5]. Although these methods improve scheduling decisions, limited attention has been given to the efficient handling of pre-empted tasks after resource reallocation. As a result, the benefits of adaptive scheduling may be reduced by the overhead associated with task interruptions and restarts.

To address this issue, this paper proposes an Adaptive Pre-emption-Aware Hierarchical Scheduling with Recovery Queues for Hadoop YARN. The proposed framework introduces a hierarchical scheduling structure that adapts resource allocation decisions according to workload characteristics and system conditions. In addition, a Recovery Queue mechanism is incorporated to retain interrupted tasks and reintroduce them into the scheduling process when resources become available. Instead of restarting pre-empted tasks from the beginning, the proposed approach aims to minimize execution loss and reduce the performance penalties associated with task interruption. By combining adaptive scheduling and recovery-oriented task management, the framework seeks to improve resource utilization, reduce makespan, and enhance overall cluster performance.

The main contributions of this work are summarized as follows:

- An adaptive hierarchical scheduling framework for Hadoop YARN capable of responding to dynamic workload variations and heterogeneous resource conditions.
- A pre-emption-aware resource management mechanism that improves responsiveness without excessively penalizing interrupted jobs.
- A Recovery Queue-based task resumption strategy that reduces execution loss associated with scheduler-induced pre-emption.
- A workload classification and prioritization approach for improving resource allocation decisions across competing jobs.
- A unified scheduling architecture that integrates adaptive scheduling, pre-emption management, and recovery support for enhanced Hadoop YARN performance.

The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 describes the proposed scheduling framework and Recovery Queue architecture. Section 4 discusses the experimental setup. Section 5 presents the results and discussion, and Section 6 concludes the paper with future research directions.

2. Related Work

Recent advances in task scheduling research have focused on improving resource utilization, reducing execution time, minimizing energy consumption, and satisfying Quality of Service (QoS) requirements in distributed computing environments. A recent systematic literature review of cloud scheduling techniques highlighted resource utilization, reliability, makespan, energy consumption, cost, and response time as the dominant optimization objectives in modern scheduling research, while also identifying the need for more adaptive and intelligent scheduling frameworks [6]. Several studies have employed artificial intelligence, reinforcement learning, predictive analytics, and metaheuristic optimization to improve scheduling decisions under dynamic workload conditions.

Machine learning and reinforcement learning have emerged as promising approaches for adaptive scheduling. LarS introduced a large language model-assisted scheduling framework that combines deep reinforcement learning and language models to optimize response time, success rate, and execution cost in cloud environments [7]. [8] proposed

an energy-aware cloud scheduling framework by integrating Hybrid Cuckoo Search with Transformer models to optimize task allocation and resource utilization. However, the approach does not support Hadoop YARN-specific scheduling, pre-emption management, checkpoint preservation, or recovery-aware execution. Reinforcement learning-based task allocation was further explored through the QMTSF framework, which employed Q-learning and upper-confidence-bound strategies to improve makespan and average processing time [9]. IntelliScheduler proposed a hybrid actor–critic learning framework for adaptive task deployment in edge–cloud systems and demonstrated improvements in operational cost and quality of experience [10]. Similarly, SLA-aware deep reinforcement learning models have been developed to reduce SLA violations, latency, and energy consumption under heterogeneous workloads [11]. A DRL-based multi-objective scheduling framework was also proposed to jointly optimize latency, energy consumption, and SLA compliance in edge-cloud environments [12].

Predictive scheduling approaches have also received significant attention. An adaptive multi-cloud scheduling framework that combines Long Short-Term Memory (LSTM) forecasting with memory-based metaheuristic optimization improved energy efficiency, reduced delays, and increased deadline satisfaction by predicting future workload behaviour [13]. ADWEH integrated dynamic prioritization with enhanced Harris Hawk Optimization to improve resource utilization, scalability, and workflow execution efficiency [14]. Similarly, workflow scheduling based on AI-driven modeling and enhanced Wild Horse Optimization improved reliability while maintaining end-to-end delay constraints in cloud environments [15].

Metaheuristic optimization remains one of the most extensively explored research directions. Quantum-inspired optimization techniques have been applied to multi-cloud scheduling to reduce latency, cost, energy consumption, and makespan through improved global search capability [16]. Modified Wombat Optimization-based scheduling approaches have demonstrated improvements in task completion time and execution cost while maintaining QoS requirements [17]. Similarly, hyper-heuristic scheduling frameworks based on Bacterial Foraging Optimization have been proposed to improve energy efficiency, resource utilization, SLA compliance, and operational cost in cloud data centers [18]. Hybrid Differential Evolution techniques have achieved superior makespan performance compared to traditional heuristic and optimization-based scheduling methods [19]. CHPSO combined heuristic load balancing with particle swarm optimization to improve resource utilization and execution efficiency in cloud environments [20]. Dwarf Mongoose Optimization-based scheduling also demonstrated effective makespan reduction and balanced resource allocation under large-scale workloads. Several researchers have investigated workflow-oriented scheduling mechanisms. WS-SSA employed the Salp Swarm Algorithm to optimize workflow execution by reducing makespan and energy consumption in heterogeneous cloud environments [21]. Energy-constrained workflow scheduling using ERWS improved makespan under strict energy budgets by intelligently allocating task-level energy resources in DVFS-enabled systems [22]. Comparative studies involving FCFS, Round Robin, Min-Min, and Max-Min scheduling approaches highlighted the impact of workflow characteristics on scheduling effectiveness and makespan optimization [23]. Energy Management Algorithm-based scheduling has also demonstrated improved workflow execution performance compared to traditional scheduling approaches such as FCFS and SJF [24].

Resource-aware and priority-aware scheduling approaches have also been widely investigated. In Hadoop environments, a Frugality Index-based scheduling framework was proposed to improve workload allocation in heterogeneous single-board-computer clusters by accounting for node capabilities and employing adaptive resource allocation policies [25]. User-priority-based scheduling and load balancing mechanisms improved resource utilization and service quality by considering user requirements and workload priorities during task allocation [26]. Hybrid machine learning frameworks integrating optimization, deep learning, and resource allocation strategies have demonstrated improvements in response time, resource utilization, and scheduling efficiency [27]. Similar resource allocation and security-oriented scheduling mechanisms have also been proposed to jointly address task scheduling and resource management challenges in cloud environments [28].

Recent studies have further extended scheduling research to cloud-fog, edge-cloud, and cloud-native environments. A dynamic scheduling framework based on quantum-inspired biased optimization improved makespan, energy efficiency, and load balancing in IoT fog-cloud environments [29]. Energy-efficient cloud-fog scheduling combined reinforcement learning and resource allocation techniques to reduce cost, delay, and schedule length [30]. Fine-grained task scheduling mechanisms based on hybrid ant colony optimization have also been proposed for intelligent edge-cloud infrastructures to improve delay and energy efficiency [31]. Furthermore, reinforcement learning-based orchestration frameworks have demonstrated the potential of intelligent scheduling for latency-sensitive microservices in Kubernetes clusters [32].

Despite the significant progress achieved by existing studies, most approaches focus primarily on optimizing scheduling objectives such as makespan, energy consumption, latency reduction, SLA compliance, resource utilization, or workflow execution efficiency. Limited attention has been paid to scheduler-induced task interruptions and the efficient management of preempted tasks. Existing approaches generally assume that interrupted tasks can be restarted or rescheduled without explicitly addressing the execution loss caused by preemption. Moreover, none of the reviewed

studies provides a dedicated recovery queue mechanism integrated with an adaptive hierarchical scheduler within the Hadoop YARN ecosystem. These limitations motivate the development of the proposed Adaptive Pre-emption-Aware Hierarchical Scheduling with Recovery Queues for Hadoop YARN.

3. Methodology

Despite significant progress in cloud and Hadoop scheduling research, existing approaches still struggle with dynamic workloads, heterogeneous resource demands, and frequent resource contention. Traditional Hadoop YARN schedulers primarily focus on resource allocation and queue management, while limited attention has been given to the impact of task interruption and recovery during pre-emption. In many situations, high-priority jobs must wait for resources to become available, whereas interrupted tasks often experience additional delays when re-entering the scheduling process.

To address these limitations, this work proposes an Adaptive Pre-emption-Aware Hierarchical Scheduling framework integrated with a Recovery Queue mechanism. The proposed approach combines workload-aware scheduling, intelligent resource allocation, controlled pre-emption, and recovery-oriented task management to improve overall scheduling efficiency and reduce the overhead of interrupted task execution.

3.1. Framework Overview

The proposed Adaptive Pre-emption-Aware Hierarchical Scheduling with Recovery Queues (APHRQ) framework is designed to improve scheduling efficiency, resource utilization, and execution reliability in Hadoop YARN environments. The framework integrates adaptive workload prioritization, hierarchical queue-based scheduling, controlled pre-emption, checkpoint-assisted recovery, and protected execution to overcome the limitations of conventional Hadoop schedulers. Unlike existing scheduling approaches that rely primarily on static scheduling policies or require interrupted tasks to restart execution, the proposed framework dynamically reallocates resources while preserving execution progress through a dedicated Recovery Queue mechanism.

Initially, all incoming jobs are analysed and assigned an Adaptive Score based on multiple scheduling attributes, including job priority, waiting time, pre-emption history, recovery status, and remaining execution time. The computed Adaptive Score determines the scheduling priority of each job and is continuously updated throughout execution to reflect changes in system conditions. Based on these scores, jobs are managed through a hierarchical queue structure comprising the Ready Queue, Recovery Queue, and Protected Queue, enabling adaptive scheduling decisions and efficient resource allocation. The framework operates through four major stages:

- Workload Classification and Adaptive Score Computation
- Adaptive Hierarchical Queue-Based Scheduling
- Pre-emption-Aware Resource Reallocation
- Recovery Queue-Based Rescheduling

During execution, high-priority jobs may pre-empt currently running workloads whenever the adaptive pre-emption conditions are satisfied. Instead of discarding the execution progress of interrupted jobs, the scheduler preserves their execution state through checkpoint-assisted recovery and transfers them to the Recovery Queue. Furthermore, resumed jobs are temporarily placed in the Protected Queue to prevent repeated interruptions and to allow meaningful progress toward execution before becoming eligible for subsequent pre-emption decisions. Overall, the proposed APHRQ framework aims to improve responsiveness for high-priority workloads while simultaneously reducing redundant re-computation, improving recovery efficiency, maintaining fair resource allocation, and minimizing starvation in dynamic Hadoop YARN environments.

3.2. Workload Classification and Priority Estimation

When jobs arrive at the Hadoop YARN environment, they are initially analyzed to determine their scheduling importance. Since different jobs may have different priority levels and varying waiting times, allocating resources solely by arrival order can degrade the responsiveness of critical workloads. Furthermore, interrupted jobs often suffer repeated delays after being returned to the waiting queue. To address these issues, the proposed framework computes an Adaptive Score for each job by considering its priority, waiting time, pre-emption history, and recovery status. The Adaptive Score, AS_i, of a job J_i is calculated using the Equation (1).

$$AS_i = P_i + \alpha Pr_i + \beta W_i + \gamma RQ_i + \frac{\eta}{Rem_i} \quad (1)$$

where, P_i is the priority value assigned with the job, Pr_i is the number of previous pre-emptions, W_i waiting time of the job, RQ_i is the recovery queue indicator, Rem_i is the remaining execution time, and α, β, γ, and η are weighting coefficients. The Recovery Queue indicator is defined as given in Equation (2).

$$RQ_i = \begin{cases} 1, & \text{when } J_i \text{ job is in Queue} \\ 0, & \text{Otherwise} \end{cases} \quad (2)$$

In the proposed framework, the adaptive scheduling score is computed by integrating multiple scheduling attributes that collectively determine each job's execution priority. The priority term preserves the relative importance of individual workloads, while the pre-emption history component gradually increases the scheduling preference of previously interrupted jobs, thereby reducing repeated postponement. The waiting time component acts as an ageing mechanism that progressively promotes jobs experiencing longer queue delays, minimizing the likelihood of starvation. The recovery queue indicator gives checkpointed jobs higher priority, allowing interrupted tasks to resume execution without repeatedly competing with newly arriving jobs. Furthermore, the remaining execution time component slightly favors jobs nearing completion, thereby improving scheduler responsiveness and overall execution efficiency.

The weighting coefficients $'\alpha', '\beta', '\gamma'$, and $'\eta'$ regulate the relative contribution of pre-emption history, waiting time, recovery status, and remaining execution time, respectively. These coefficients were empirically selected through simulation experiments to achieve a balanced trade-off between scheduling responsiveness, fairness, recovery efficiency, and execution stability under heterogeneous Hadoop YARN workloads. For all experiments reported in this work, the coefficients were fixed at $\alpha = 0.50,\ \beta = 0.05,\ \gamma = 2.50$ and η=10.0 . These values were determined empirically after multiple simulation trials and were retained throughout the evaluation to ensure consistent comparison across all schedulers.

Consequently, the computed Adaptive Score provides a dynamic priority ranking for all waiting jobs, allowing the scheduler to continuously adjust scheduling decisions according to changing workload characteristics. By jointly considering job priority, waiting time, pre-emption history, recovery status, and remaining execution time, the proposed APHRQ scheduler improves scheduling responsiveness, supports efficient checkpoint-based recovery, and promotes fair resource allocation under heterogeneous Hadoop YARN workloads. The computed Adaptive Score is subsequently used for hierarchical queue assignment and pre-emption decisions in the proposed scheduling framework.

3.3. Adaptive Hierarchical Queue-Based Scheduling

After computing the adaptive scores, jobs are organized into a hierarchical queue structure consisting of the Ready Queue, Recovery Queue, and Protected Queue. During each scheduling cycle, the scheduler dynamically re-evaluates the priority of all eligible jobs according to their Adaptive Score. Unlike conventional Hadoop YARN schedulers that primarily rely on static scheduling policies, the proposed APHRQ scheduler continuously updates scheduling decisions in response to changes in waiting time, recovery status, and pre-emption history. Consequently, the scheduler always selects the most appropriate job for execution according to Equation (3).

$$J_{selected} = \arg\max_{J_i \in Q}(AS_i) \quad (3)$$

here, $J_{selected}$ is the job selected for execution, AS_i is the adaptive score of job J_i and Q is as defined in Equation (4).

$$Q = \begin{cases} Protected\ Queue & if\ nonempty \\ Ready\ Queue \cup Recovery\ Queue, & otherwise \end{cases} \quad (4)$$

The Adaptive Score is recalculated whenever a new job arrives, a running job completes, a pre-emption event occurs, or a job resumes from the Recovery Queue. Consequently, jobs experiencing prolonged waiting, repeated interruptions, or checkpoint-based recovery gradually receive higher scheduling preference, enabling the scheduler to adapt dynamically to changing workload conditions. To prevent repeated interruptions, jobs resumed from the Recovery Queue are temporarily transferred to the Protected Queue. Jobs in the Protected Queue are exempt from immediate re-pre-emption, allowing them to execute for a sufficient duration before becoming eligible for subsequent scheduling decisions. This mechanism preserves checkpoint progress, reduces redundant execution, and improves recovery efficiency while maintaining scheduling fairness.

3.4. Pre-emption-Aware Resource Reallocation

In dynamic Hadoop YARN environments, high-priority jobs may arrive while system resources are occupied by currently executing workloads. Instead of indiscriminately interrupting running jobs, the proposed framework performs controlled pre-emption based on adaptive score evaluation and execution progress. This strategy ensures that pre-emption occurs only when it provides a measurable scheduling benefit, thereby reducing unnecessary context switching and execution overhead.

Let AS_c represent the adaptive score of a candidate waiting job and AS_r represent the adaptive score of the currently running job. Pre-emption is triggered only when the condition in Equation (5) is satisfied.

$$AS_c > AS_r + \delta \wedge Rem_c < Rem_r \quad (5)$$

The threshold δ ensures that a running job is interrupted only when the incoming job demonstrates significantly higher

scheduling importance. This prevents frequent context switching caused by minor differences in adaptive scores. When pre-emption occurs, the execution state of the interrupted job is preserved and transferred to the Recovery Queue. If the number of pre-emptions experienced by a job reaches the predefined threshold, the job is temporarily moved to the Protected Queue, where further pre-emption is disabled until meaningful execution progress is achieved. This mechanism prevents repeated interruptions, improves execution stability, and reduces the likelihood of starvation. During checkpoint creation, execution progress, elapsed execution time, remaining execution time, and checkpoint information are retained to support future resumption. Since jobs stored in the Recovery Queue receive additional preference through the recovery indicator RQ_i, they are more likely to resume execution during subsequent scheduling cycles, thereby reducing repeated postponement and enabling efficient checkpoint-based recovery. By integrating adaptive score-based pre-emption, checkpoint-assisted recovery, and Protected Queue management, the proposed APHRQ scheduler improves responsiveness to urgent workloads while minimizing redundant execution, reducing repeated interruptions, and maintaining execution stability in dynamic Hadoop YARN environments.

3.5. Recovery Queue Mechanism

The Recovery Queue is the primary architectural contribution of the proposed APHRQ framework. In conventional Hadoop YARN schedulers, interrupted jobs are typically returned to the normal waiting queue, where they must again compete with newly arriving workloads. Consequently, partially executed jobs may experience repeated postponement, increased waiting time, and unnecessary recomputation due to successive interruptions.

To address this limitation, every pre-empted job is transferred to a dedicated Recovery Queue, where its execution state is preserved through checkpoint-assisted recovery. The scheduler maintains essential execution information, including the job identifier, execution progress, elapsed execution time, remaining execution time, checkpoint information, and pre-emption history. When computational resources become available, jobs residing in the Recovery Queue are re-evaluated using the Adaptive Score defined in Equation (1). The recovery indicator RQ_i assigns additional scheduling preference to checkpointed jobs, enabling faster execution resumption while reducing repeated postponement and redundant re-computation.

Furthermore, jobs resumed from the Recovery Queue are temporarily transferred to the Protected Queue, where immediate re-pre-emption is disabled until meaningful progress in execution is achieved. This mechanism prevents repeated interruptions, preserves checkpoint progress, and improves execution stability without compromising scheduling responsiveness. By integrating checkpoint-assisted recovery with adaptive scheduling and controlled pre-emption, the proposed APHRQ framework significantly reduces redundant execution, improves fairness, minimizes recovery overhead, and enhances overall scheduling efficiency in dynamic Hadoop YARN environments.

The individual components of the proposed APHRQ framework operate collaboratively to support adaptive resource allocation, controlled pre-emption, checkpoint-assisted recovery, and starvation-aware scheduling. While the preceding subsections describe each component independently, the overall scheduling behaviour is governed by a unified decision-making process. To illustrate how workload classification, adaptive queue management, resource allocation, pre-emption, checkpoint recovery, and protected execution are integrated during runtime, the complete scheduling procedure of the proposed framework is presented in Algorithm 1.

Algorithm 1: Adaptive Pre-emption-Aware Hierarchical Scheduling with Recovery Queue (APHRQ)

Input: Job set $J = \{J_1, J_2, J_3, ..., J_n\}$, where each job J_i contains arrival time AT_i, execution time ET_i, priority P_i, and queue type Q_i.

Output: Completed job schedule with adaptive pre-emption, checkpointing and recovery support

```
Initialize Ready Queue, Recovery Queue, Protected Queue and Completed List
Set current time t ← 0
while (unfinished jobs exist) do
    Insert newly arrived jobs into Ready Queue
    for each waiting job J_i in Ready Queue or Recovery Queue do
        Compute Adaptive Score, AS_i
    end for
    if a job is running and APHRQ fair-share quantum is reached, then
        if other jobs are waiting and running job is not protected, then
            Move the running job back to the Ready Queue
        end if
    end if
```

```
        if (running job exists) then
            Select the candidate job J_C from Ready Queue with the
highest Adaptive Score
            if AS_c > AS_r + δ, and RT_c < RT_r, and (running job not protected), then
                Save execution progress
                Move running job to Recovery Queue
                Increase pre-emption count
                if pre-emption count reaches threshold, then
                    Move running job to Protected Queue
                End if
                Start execution of J_C
            end if
        end if
        if (no running job exists) then
            if (Protected Queue not empty) then
                Select the highest-score job from the Protected Queue
            else
                Combine Ready Queue and Recovery Queue
                Select the highest-score job from the Combined Queue
                if selected job belongs to Recovery Queue, then
                    Restore checkpoint and resume execution
                end if
            end if
        end if
        Execute selected job
        Update executed time, remaining time, progress and resource usage
        if (job completed) then
            Move job to Completed List
        end if
        t ← t + 1
    end while
    Return scheduling results
```

The proposed APHRQ framework integrates adaptive priority estimation, controlled pre-emption, and Recovery Queue-based task resumption within a unified Hadoop YARN scheduling environment. By reducing unnecessary task restarts and improving the handling of interrupted workloads, the framework aims to enhance overall scheduling efficiency and resource utilization.

4. Experimental Section

The proposed Adaptive Pre-emption-Aware Hierarchical Scheduling with Recovery Queues (APHRQ) framework was evaluated using a simulation-based Hadoop YARN scheduling environment developed in Python. The simulator was designed to emulate the behaviour of multiple scheduling policies under varying workload conditions while supporting adaptive scheduling, pre-emption handling, checkpoint preservation, and Recovery Queue-based task resumption.

To provide both behavioral validation and scalability assessment, the experiments were conducted in two stages. The first stage focused on visual demonstration using a small workload consisting of six jobs, enabling detailed observation of scheduling decisions, queue transitions, pre-emption events, checkpoint creation, and recovery operations. The second stage evaluated scalability using larger workloads containing 50, 100, and 500 jobs to analyze the effectiveness of the proposed framework under increasing scheduling pressure.

The APHRQ scheduler was compared against commonly used scheduling approaches including FIFO, Fair Scheduler, Capacity Scheduler, Shortest Job First (SJF), Shortest Remaining Time First (SRTF), and a pre-emption-based scheduler without recovery support. All schedulers were executed under identical workload conditions to ensure fair comparison.

4.1. Simulation Environment

The experimental framework was implemented using Python and Streamlit. The simulation engine models job arrivals, queue management, resource allocation, pre-emption decisions, checkpoint preservation, recovery operations, and job completion events. Interactive visualization components were incorporated to observe scheduler behavior during execution. Table 1, below, shows the parameters of the simulation environment.

Table 1: Simulation Environment

Parameter	Value
Implementation Language	Python
Framework	Streamlit
Scheduling Environment	Hadoop YARN Simulation
Execution Modes	Visual Demonstration, Scalability Evaluation
Compared Schedulers	FIFO, Fair, Capacity, SJF, SRTF, Pre-emption without Recovery, APHRQ
Workload Types	Normal, Pre-emption Heavy, Critical
Random Seed	42
Scalability Levels	50, 100, 500 Jobs
Checkpoint Support	APHRQ Only
Protected Queue	APHRQ Only

Figure 1 presents the Streamlit-based simulation interface developed for evaluating the proposed APHRQ framework. The dashboard supports scheduler selection, workload visualization, and execution control for both visual demonstration and scalability evaluation modes.

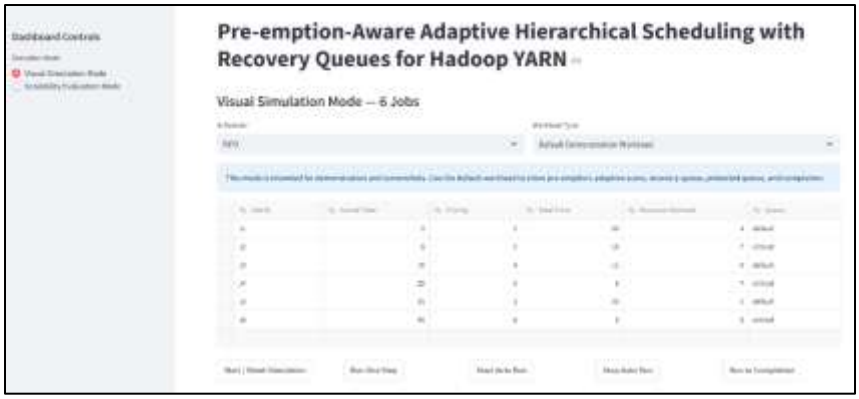

Figure 1: Streamlit-based simulation dashboard developed for evaluating the proposed APHRQ scheduler.

4.2. Workload Configuration and Visual Demonstration

To validate the operational behavior of the proposed scheduler, a visual demonstration mode with six representative jobs was used. The workload includes jobs with varying arrival times, execution durations, priorities, and resource demands to intentionally trigger preemption and recovery scenarios. This configuration enables detailed observation of queue transitions, adaptive scheduling decisions, checkpoint preservation, and task-resumption mechanisms before large-scale scalability experiments.

During execution, jobs dynamically move among the Ready Queue, Running Queue, Recovery Queue, Protected Queue, and Completed Queue based on scheduling decisions. Figure 2 presents a representative execution snapshot illustrating the runtime state of the proposed APHRQ framework and shows how interrupted jobs are transferred to the Recovery Queue while active jobs continue execution, demonstrating the framework's ability to manage workload transitions without losing execution progress.

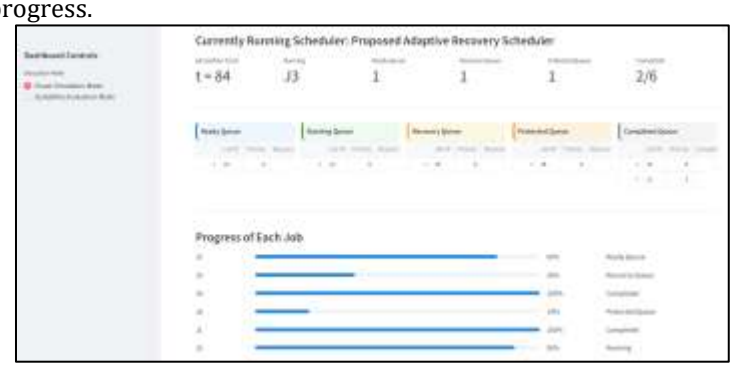

Figure 2: Runtime queue transitions and execution progress during APHRQ scheduling

The workload characteristics used in the simulation are summarized in Table 2.

Table 2: Workload Characteristics

Workload	Description
Normal	Random arrivals and execution times
Pre-emption Heavy	Long, low-priority jobs followed by urgent jobs
Critical	Bursts of high-priority jobs

The APHRQ scheduler continuously evaluates job characteristics before making scheduling decisions. Figure 3 illustrates the adaptive decision information generated during execution, including priority level, waiting time, pre-emption count, queue status, and adaptive score. These parameters collectively influence job selection, resource allocation, and pre-emption decisions within the proposed framework.

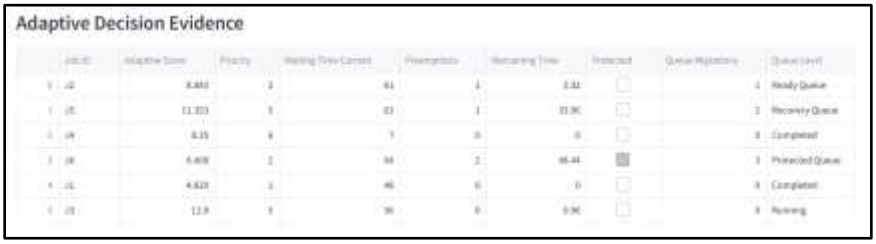

Figure 3: Adaptive scheduling decision information generated during runtime

Whenever a running job is interrupted, its execution state is preserved before being transferred to the Recovery Queue. Figure 4 presents the checkpoint information maintained by the scheduler, including execution progress and remaining execution requirements. This mechanism enables interrupted jobs to resume execution from their previously saved state, thereby reducing redundant computation and improving scheduling efficiency.

Figure 4: Checkpoint preservation and Recovery Queue information for interrupted jobs

Figure 5 presents a representative snapshot of the queue status log generated during APHRQ execution. The log records queue transitions, job execution events, pre-emption decisions, checkpoint creation, and Recovery Queue operations throughout the scheduling process. The recorded events provide additional evidence of the interaction between adaptive scheduling, controlled pre-emption, and recovery-aware execution.

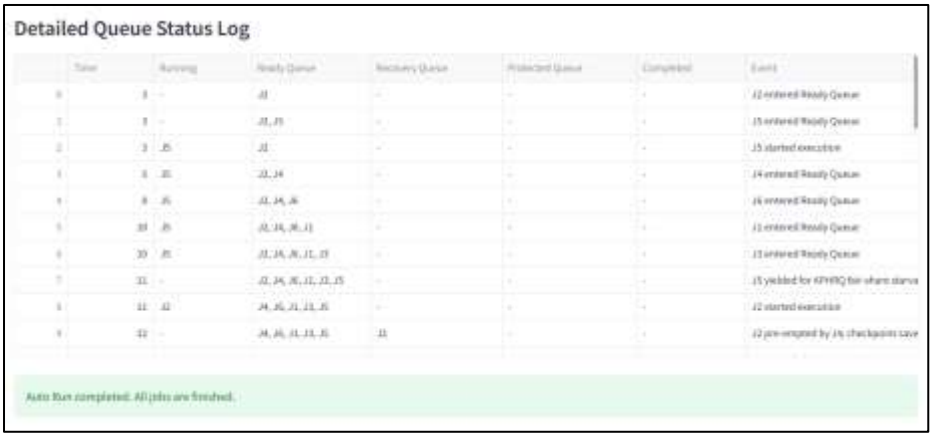

Figure 5: Detailed queue status log illustrating scheduling events and Recovery Queue operations

4.2.1. Representative Execution Scenario

To illustrate the behavior of the proposed APHRQ framework, a representative six-job execution scenario was analyzed.

During execution, low-priority running jobs were pre-empted whenever higher-priority workloads required immediate resource allocation. Instead of restarting interrupted jobs from the beginning, the
scheduler preserved their execution state using checkpoint-assisted recovery and transferred them to the Recovery Queue. When computational resources became available, these partially executed jobs resumed from their previously saved checkpoints, thereby reducing redundant re-computation and improving overall scheduling efficiency. This behavior can be observed in the queue transitions, checkpoint information, and execution timeline presented in Figures 2–5.

The representative six-job execution confirms the correctness of the proposed scheduling workflow and illustrates the interaction between adaptive scheduling, controlled pre-emption, checkpoint-assisted recovery, and protected execution. Following this qualitative validation, the proposed framework is quantitatively evaluated using large-scale workloads of 50, 100, and 500 jobs, and the results are presented in the next section.

4.3. Scalability Evaluation Setup

Following the visual validation stage, scalability experiments were conducted using workloads containing 50, 100, and 500 jobs. The scalability mode was designed to evaluate scheduler behavior under increasing workload intensity without animation overhead. This mode executes large workloads without graphical animation while collecting aggregate scheduling metrics for comparative performance evaluation, enabling consistent comparison among FIFO, Fair Scheduler, Capacity Scheduler, SJF, SRTF, Pre-emption without Recovery, and the proposed APHRQ scheduler. Figure 6 illustrates the scalability evaluation interface used for configuring workload size, workload type, and experiment parameters.

Figure 6: Scalability evaluation interface for large-scale workload analysis

All experiments were repeated using identical workload configurations and a fixed random seed to ensure reproducibility and eliminate experimental bias. For each workload category and scheduling algorithm, the simulator collected performance metrics including average waiting time, turnaround time, makespan, throughput, resource utilization, fairness indices, starvation metrics, checkpoint statistics, and recovery overhead. The resulting performance comparisons are presented in the following section.

4.4. Performance Evaluation Metrics

The performance of the proposed APHRQ framework was evaluated using a set of scheduling metrics that capture execution efficiency, resource utilization, fairness, and the effectiveness of recovery-aware scheduling. These metrics were computed for different workload scenarios and were subsequently used to compare APHRQ with FIFO, Fair Scheduler, Capacity Scheduler, SJF, SRTF, and Pre-emption without Recovery. Since the proposed framework was evaluated using a discrete-event simulation, all time-related performance metrics, including waiting time, turnaround time, saved work, wasted work, and starvation severity, are reported in Simulation Time Units (STU) rather than real execution time.

4.4.1. Makespan

Makespan represents the total time required to complete all submitted jobs. It is commonly used to evaluate the overall efficiency of a scheduling algorithm. Lower makespan values indicate faster completion of the workload and better utilization of available resources. It is computed as shown in Equation (6).

$$Makespan = \max(C_i) \tag{6}$$

where C_i denotes the completion time of job i.

4.4.2. Average Waiting Time

Average Waiting Time (AWT), as given in Equation (7), measures the duration jobs remain in scheduling queues before receiving execution resources. Lower waiting times indicate better responsiveness of the scheduler.

$$AWT = \frac{\sum_{i=1}^{n} WT_i}{n} \tag{7}$$

where WT_i represents the waiting time of job i.

4.4.3. Average Turnaround Time

Average Turnaround Time (ATT) is the total time elapsed from job arrival to job completion. This metric reflects the overall efficiency of submitted workloads and is calculated using Equation (8).

$$ATT = \frac{\sum_{i=1}^{n} TAT_i}{n} \tag{8}$$

where TAT_i represents the turnaround time of job i.

4.4.4. Throughput

Throughput, shown in Equation (9), indicates the number of jobs completed during the execution period. Higher throughput values represent improved scheduling efficiency.

$$Throughput = \frac{N}{T} \tag{9}$$

Where N is the total number of completed jobs and T is the total execution time.

4.4.5. Resource Utilization

Resource Utilization (RU), as given in Equation (10), measures the effectiveness with which available computational resources are used during execution.

$$RU = \frac{\sum ResourceUsage_t}{ClusterCapacity\ X\ SimulationTime} \text{X } 100 \tag{10}$$

here, $ResourceUsage_t$ is the resource consumption at time t and $ClusterCapacity$ is the total available cluster capacity. Higher utilization values indicate more effective use of cluster resources.

4.4.6. Pre-emption Count

Pre-emption Count (PC) represents the total number of interruptions performed during scheduling. This metric is used to evaluate the scheduler's behaviour under dynamic workload conditions. This is computed as given in Equation (11).

$$PC = \sum_{i=1}^{n} Pr_i \tag{11}$$

where Pr_i denotes the number of pre-emptions experienced by job i.

4.4.7. Saved Work and Wasted Work

Saved Work represents the amount of execution progress preserved through the Recovery Queue mechanism, whereas Wasted Work denotes the execution effort lost due to interruptions. These are represented by Equations (12 - 14) respectively.

$$SavedWork = \sum_{i=1}^{n} CheckpointProgress_i \tag{12}$$

$$WastedWork = \sum_{i=1}^{n} RestartedExecution_i \tag{13}$$

$$CheckpointProgress_i = ET_i^{saved}; RestartedExecution_i = ET_i^{discarded} \tag{14}$$

where, $CheckpointProgress_i$ denotes the execution time of the job J_i preserved at the checkpoint before pre-emption, $RestartedExecution_i$ denotes the execution time discarded when a pre-empted job is restarted without recovery, ET_i^{saved} is denotes the amount of execution time already completed and preserved at the checkpoint before the job J_i is pre-empted and $ET_i^{discarded}$ denotes the amount of execution time lost when a pre-empted job is restarted from the beginning without checkpoint recovery. Higher Saved Work and lower Wasted Work values indicate better recovery efficiency. These metrics help quantify the effectiveness of checkpoint preservation and recovery-aware execution.

4.4.8. Re-computation Reduction

Re-computation Reduction (RR) quantifies the percentage of execution effort preserved through checkpoint-assisted recovery. It measures the extent to which the proposed Recovery Queue mechanism avoids redundant re-execution after job pre-emption. Higher Re-computation Reduction values indicate more effective recovery support and lower computational overhead. This metric is computed using Equation (15).

$$RR(\%) = \frac{SavedWork}{SavedWork + WastedWork} \text{ X } 100 \quad (15)$$

where $SavedWork$ represents the execution progress preserved through checkpoint-assisted recovery, and $WastedWork$ denotes the execution effort lost due to task interruption and restart. Larger values of Re-computation Reduction indicate greater preservation of completed work and improved recovery efficiency.

4.4.9. Jain Fairness Index

The Jain Fairness Index (JFI) is used to evaluate how fairly resources are distributed among competing jobs. It is as shown in Equation (16).

$$JFI = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n * \sum_{i=1}^{n} x_i^2} \quad (16)$$

where x_i represents the normalized CPU service (or execution progress) by job i. Values closer to 1 indicate better fairness.

4.4.10. Recovery-Aware Fairness Index (RAFI)

The Recovery-Aware Fairness Index (RAFI) extends the conventional Jain Fairness Index by incorporating the effect of checkpoint-assisted recovery. This metric evaluates both fair resource allocation and recovery efficiency during scheduling. Higher RAFI values indicate better fairness while effectively preserving execution progress through the Recovery Queue mechanism. It is computed using Equation (17).

$$RAFI = JFI \text{ X } \left(1 - \frac{WastedWork}{SavedWork + WastedWork + \varepsilon}\right) \quad (17)$$

where, ε is a small positive constant $\left(10^{-9}\right)$ introduced to avoid division by zero when no checkpoint activity is recorded.

4.4.11. Starved Jobs

Starved Jobs represent workloads that experience excessive waiting before receiving execution resources. This is computed using Equations (18) and (19). In this study, a job is considered starved when its waiting time exceeds twice the average waiting time of the workload.

$$SJ = \sum_{i=1}^{n} Starved_i \quad (18)$$

where,

$$Starved_i = \begin{cases} 1, & WT_i > 2 \; X \; AWT \\ 0, & otherwise \end{cases} \quad (19)$$

Lower values indicate better fairness and more balanced resource allocation.

4.4.12. Starvation Severity Score

The Starvation Severity Score (SSS) quantifies the cumulative waiting delay experienced by starved jobs beyond the starvation threshold, thereby reflecting the overall severity of starvation during scheduling as shown in (20).

$$SSS = \sum_{i=1}^{n} Starved_i \; X \; WT_i \quad (20)$$

The above metrics collectively evaluate scheduler performance from the perspectives of execution efficiency, resource utilization, recovery effectiveness, fairness, starvation resilience, and scalability. Together, they provide a comprehensive assessment of the proposed APHRQ framework under diverse Hadoop YARN workload scenarios.

5. Results and Discussions

This section presents an experimental evaluation of the proposed Adaptive Pre-emption-Aware Hierarchical Scheduling with Recovery Queues (APHRQ) framework, compared with FIFO, Fair Scheduler, Capacity Scheduler, SJF, SRTF, and Pre-emption without Recovery (PWR). The evaluation aims to demonstrate the effectiveness of integrating adaptive scheduling with checkpoint-assisted recovery for improving scheduling efficiency under dynamic Hadoop YARN workloads. Unlike conventional schedulers that restart interrupted tasks or rely on static scheduling decisions, APHRQ preserves execution progress via the Recovery Queue and dynamically adapts its scheduling decisions to changing workload conditions. The experimental results presented in Table 3 demonstrate the impact of the proposed framework on scheduling performance, recovery efficiency, fairness, starvation resilience, and scalability across

workloads comprising 50, 100, and 500 jobs.

Different workload categories were intentionally employed to evaluate specific characteristics of the proposed scheduler. The Normal workload was used to assess general scheduling efficiency through conventional performance metrics such as waiting time, turnaround time, throughput, and resource utilization. The Pre-emption Heavy workload was designed to evaluate the effectiveness of the proposed Recovery Queue mechanism by analyzing pre-emption count, saved work, wasted work, and re-computation reduction. The Critical workload was used to investigate fairness and starvation behavior under high-priority burst arrivals using Jain Fairness Index, Recovery-Aware Fairness Index, Starved Jobs, and Starvation Severity Score. This workload-oriented evaluation enables each component of the proposed APHRQ framework to be analyzed under the conditions for which it is specifically designed.

Table 3: Consolidated experimental results for the evaluated scheduling algorithms across different workload scenarios

Waiting Time and Turnaround Time Analysis (Normal Workload)							
Metric		Waiting Time (STU)			Turnaround Time (STU)		
Scheduler	Size	50	100	500	50	100	500
FIFO		557.98	1090.65	5789.59	582.04	1114.02	5813.42
Fair		866.3	1651.3	8716.35	890.36	1674.67	8740.18
Capacity		570.34	1089.61	5712.8	594.4	1112.98	5736.63
SJF		429.46	827.46	4293.18	453.52	850.83	4317.01
SRTF		423.1	817.17	4239.77	447.16	840.54	4263.6
PWR		656.46	1235.01	6360.09	680.52	1258.38	6383.92
APHRQ		495.74	964	5009.4	519.78	987.36	5033.22
Throughput and Resource Utilization Analysis (Normal Workload)							
Metric		Throughput (STU)			Resource Utilization (%)		
Scheduler	Size	50	100	500	50	100	500
FIFO		0.0416	0.0428	0.042	38.02	39.14	43.26
Fair		0.0416	0.0428	0.042	38.4	39.53	43.69
Capacity		0.0416	0.0428	0.042	38.78	39.92	44.12
SJF		0.0423	0.0435	0.0427	39.54	40.73	44.99
SRTF		0.0427	0.0439	0.0431	39.93	41.13	45.42
PWR		0.0373	0.0385	0.0379	34.98	36.01	39.8
APHRQ		0.0457	0.047	0.0463	42.91	44.2	48.83
Pre-emption Behavior Analysis (Preemption Heavy Workload)							
Metric		Pre-emption Count			Saved Work (STU)		
Scheduler	Size	50	100	500	50	100	500
SRTF		2	4	2	0	0	0
PWR		2	1	1	0	0	0
APHRQ		9	13	68	10.08	14.56	215.31
Re-computation Overhead Analysis (Preemption Heavy Workload)							
Metric		Wasted Work (STU)			Re-computation Reduction (%)		
Scheduler	Size	50	100	500	50	100	500
SRTF		0	0	0	0	0	0
PWR		4.6	5.52	4.6	0	0	0
APHRQ		0	0	0	100	100	100
Recovery-Aware Fairness Analysis (Critical Workload)							
Metric		Jain Fairness Index			RFAI		
Metric	Size	50	100	500	50	100	500
FIFO		0.8817	0.889	0.8972	0.4938	0.4978	0.5074
Fair		0.9245	0.9156	0.9168	0.5954	0.5127	0.6058
Capacity		0.8641	0.8627	0.8616	0.496	0.4892	0.4873
SJF		0.8631	0.863	0.8586	0.4834	0.4893	0.494
SRTF		0.8618	0.8613	0.8577	0.4826	0.4883	0.4935
PWR		0.8656	0.8629	0.8621	0.4968	0.4893	0.4876
APHRQ		0.9332	0.9178	0.9119	0.7834	0.7708	0.817
Starvation Analysis (Critical Workload)							

Metric		Starved Count			Starvation Severity Score		
Metric	Size	50	100	500	50	100	500
FIFO		10	20	96	1866.86	6565.66	157805.1
Fair		4	20	28	59.33	847.51	5680.65
Capacity		9	19	96	1712.2	6545.26	171987.6
SJF		10	19	89	2257.68	8673.18	220808.1
SRTF		10	19	89	2234.92	8555.34	217956.8
PWR		9	19	96	1889.45	7185.45	189947.3
APHRQ		8	16	52	290.84	1051.34	13814.65

5.1. Waiting Time and Turnaround Time Analysis

Reducing waiting time and turnaround time improves scheduler responsiveness and overall workload completion. Figure 7 compares the Average Waiting Time (AWT) and Average Turnaround Time (ATT) of the evaluated schedulers under the normal workload.

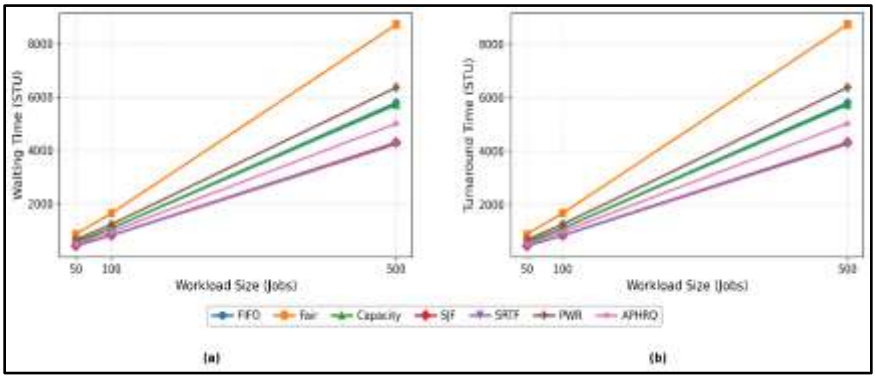

Figure 7: Comparison of scheduling algorithms based on (a) Waiting Time (WAT) and (b) Turnaround Time (TAT) across different workload sizes

SJF and SRTF achieve the lowest waiting and turnaround times by prioritizing shorter jobs. Compared with FIFO, Fair Scheduler, Capacity Scheduler, and Pre-emption without Recovery (PWR), the proposed APHRQ consistently reduces both waiting time and turnaround time across all workload sizes. At the 500-job workload, APHRQ decreases the waiting time from 6360.09 STU (PWR) to 5009.40 STU and the turnaround time from 6383.92 STU to 5033.22 STU. Although APHRQ does not attain the minimum waiting time, it achieves competitive scheduling performance while simultaneously supporting checkpoint-assisted recovery and adaptive pre-emption.

5.2. Throughput and Resource Utilization Analysis

Figure 8 illustrates the throughput and resource utilization achieved by different schedulers under the normal workload. The proposed APHRQ consistently achieves the highest throughput and resource utilization across all workload sizes. At the 500-job workload, APHRQ records a throughput of 0.0463 Jobs/STU, compared with 0.0379 Jobs/STU for PWR, while resource utilization increases from 39.80% to 48.83%. These results indicate that checkpoint-assisted recovery and adaptive resource allocation enable APHRQ to utilize cluster resources more efficiently while reducing redundant execution.

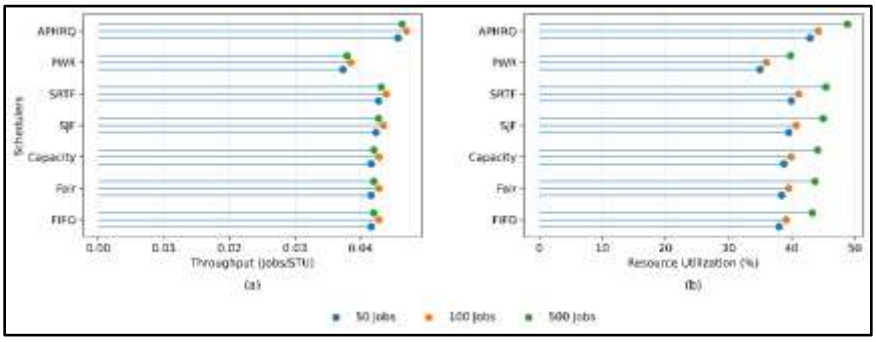

Figure 8: Comparison of scheduling algorithms based on (a) Throughput and (b) Resource Utilization (RU) across different workload sizes

Although SJF and SRTF provide competitive scheduling efficiency, they do not preserve execution progress during pre-emption. By integrating adaptive scheduling with Recovery Queue-based execution, APHRQ improves overall system productivity while minimizing redundant execution, thereby achieving better resource utilization without compromising scheduling responsiveness.

5.3. Pre-emption Behavior and Recovery Effectiveness Analysis

The pre-emption-heavy workload was used to evaluate scheduler behaviour under frequent interruption conditions. As shown in Figure 9(a), APHRQ performs more pre-emptions than SRTF and PWR because it continuously re-evaluates job priority using adaptive score-based decisions. However, these pre-emptions do not lead to re-computation loss because APHRQ preserves execution progress through checkpoint-assisted recovery. Figure 9(b) shows that SRTF and PWR record zero saved work, whereas APHRQ preserves 215.31 STU of execution work for the 500-job workload. This indicates that the novelty of APHRQ is not merely reducing the number of pre-emptions, but reducing their cost by using the Recovery Queue to resume interrupted jobs without discarding completed execution.

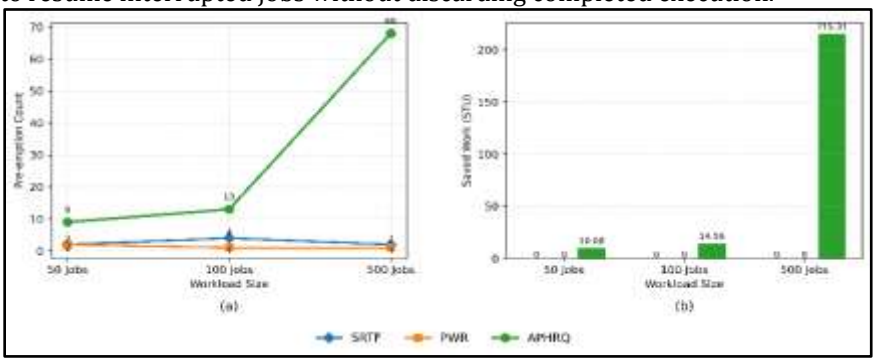

Figure 9: Comparison of (a) Pre-emption Count and (b) Saved Work under the pre-emption-heavy workload.

5.4. Re-computation Overhead Analysis

The pre-emption-heavy workload was further analyzed to evaluate the impact of checkpoint-assisted recovery on re-computation overhead. Figure 10(a) compares the amount of execution work lost due to task interruption. As expected, PWR incurs wasted work because interrupted jobs restart from the beginning, resulting in 4.60 STU, 5.52 STU, and 4.60 STU of discarded execution for the 50-, 100-, and 500-job workloads, respectively. In contrast, both SRTF and the proposed APHRQ record zero wasted work. Although SRTF avoids execution loss under the evaluated workload, it does not provide checkpoint-based recovery or recovery-aware rescheduling.

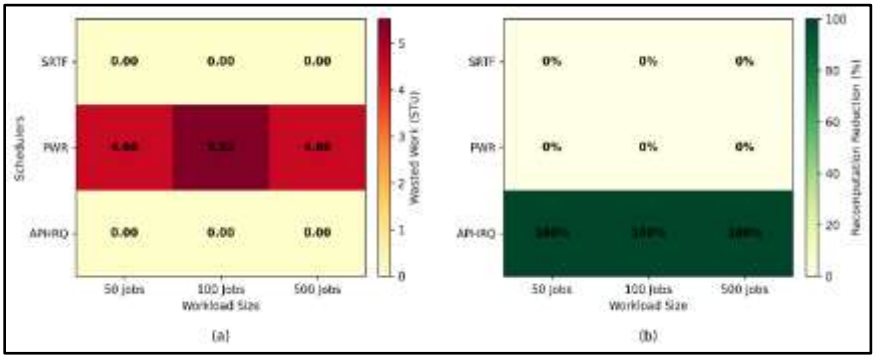

Figure 10: Comparison of (a) Wasted Work and (b) Re-computation Reduction under the pre-emption-heavy workload

Figure 10(b) presents the re-computation reduction achieved by the evaluated schedulers. APHRQ consistently achieves 100% re-computation reduction across all workload sizes because interrupted jobs resume execution from their previously saved checkpoints instead of restarting from the beginning. Consequently, previously completed execution is preserved, eliminating redundant computation during repeated pre-emption events. In contrast, neither SRTF nor PWR provides re-computation reduction since they do not employ a checkpoint-assisted Recovery Queue mechanism. These results demonstrate that the proposed APHRQ framework effectively minimizes re-computation overhead while improving execution efficiency under dynamic Hadoop YARN environments.

5.5. Fairness and Recovery-Aware Fairness Analysis

The critical workload was used to evaluate fairness under frequent arrivals of high-priority jobs. Figure 11(a) compares the Jain Fairness Index (JFI) achieved by different schedulers. The proposed APHRQ maintains fairness comparable to

the Fair Scheduler across all workload sizes while outperforming FIFO, Capacity, SJF, SRTF, and PWR. For the 500-job workload, APHRQ achieves a JFI of 0.9119, indicating balanced resource allocation despite frequent pre-emption events.

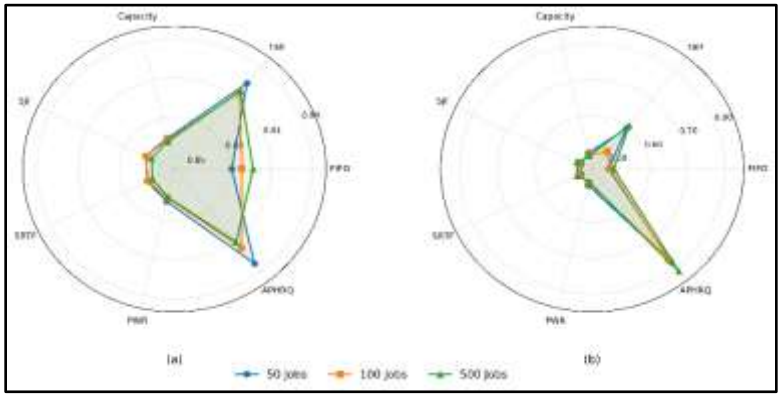

Figure 11: Comparison of conventional scheduling fairness using (a) Jain Fairness Index (JFI) and (b) Recovery-Aware Fairness Index (RAFI) under the critical workload

Figure 11(b) presents the proposed Recovery-Aware Fairness Index (RAFI), which evaluates fairness by considering the scheduling opportunities provided to interrupted jobs. Unlike conventional fairness metrics, RAFI reflects the effectiveness of recovery-aware scheduling after task pre-emption. APHRQ consistently records the highest RAFI across all workload sizes, reaching 0.8170 for the 500-job workload, whereas the closest baseline (Fair Scheduler) achieves 0.6058. This improvement is primarily attributed to the Recovery Queue and Protected Queue mechanisms, which allow interrupted jobs to resume execution without repeatedly competing with newly arriving workloads. These results demonstrate that the proposed APHRQ framework improves not only conventional scheduling fairness but also fairness for partially executed jobs, thereby reducing repeated postponement and supporting recovery-aware resource allocation in dynamic Hadoop YARN environments

5.6. Starvation and Scheduling Stability Analysis

The critical workload was further used to evaluate starvation under sustained resource contention. Figure 12(a) compares the number of starved jobs produced by each scheduler. APHRQ consistently reduces the number of starved jobs compared with FIFO, Capacity, SJF, SRTF, and PWR by combining adaptive scheduling with checkpoint-assisted recovery. For the 500-job workload, the number of starved jobs decreases from 96 in FIFO and PWR to 52 in APHRQ. Although the Fair Scheduler records the lowest starvation count for smaller workloads, APHRQ provides a better balance between starvation reduction and recovery-aware execution.

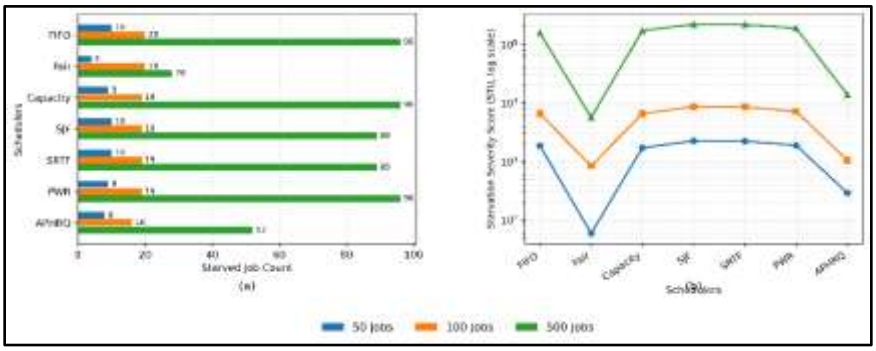

Figure 12: Comparison of (a) Starved Job Count and (b) Starvation Severity Score (SSS) under the critical workload

Figure 12(b) presents the Starvation Severity Score (SSS), which reflects both the number of starved jobs and the extent of their waiting delay. APHRQ achieves substantially lower SSS values than FIFO, Capacity, SJF, SRTF, and PWR across all workload sizes, indicating that the number of severely delayed jobs is considerably reduced. The improvement is mainly attributed to the Recovery Queue, Protected Queue, and adaptive score-based scheduling, which collectively prevent interrupted jobs from being repeatedly postponed. These results demonstrate that APHRQ improves scheduling stability while maintaining equitable resource allocation under highly dynamic Hadoop YARN workloads.

Overall, the experimental results demonstrate the effectiveness of the proposed Adaptive Pre-emption-Aware

Hierarchical Scheduling with Recovery Queues (APHRQ) framework across different workload conditions. The proposed scheduler achieves competitive scheduling performance under normal workloads while providing significant advantages during pre-emption-heavy and critical workload scenarios. The integration of adaptive scheduling, checkpoint preservation, the Recovery Queue, and the Protected Queue effectively reduces redundant re-computation, improves recovery-aware fairness, minimizes starvation, and maintains efficient resource utilization. These results confirm that APHRQ provides a balanced scheduling strategy that improves both execution efficiency and recovery-aware resource management in dynamic Hadoop YARN environments.

6. Conclusion and Future Scope

This paper presented the Adaptive Pre-emption-Aware Hierarchical Scheduling with Recovery Queues (APHRQ) framework to improve scheduling efficiency in Hadoop YARN environments. The proposed framework integrates adaptive score-based scheduling, controlled pre-emption, checkpoint-assisted recovery, a dedicated Recovery Queue, and a Protected Queue to address the limitations of conventional schedulers that either delay urgent workloads or incur redundant execution after task interruption. Extensive simulation experiments under normal, pre-emption-heavy, and critical workload conditions demonstrated that APHRQ effectively improves resource utilization, throughput, recovery efficiency, recovery-aware fairness, and starvation reduction while maintaining competitive scheduling performance. Unlike existing scheduling approaches, APHRQ minimizes the execution overhead associated with pre-emption by preserving execution progress and enabling interrupted jobs to resume from their saved checkpoints. The proposed Recovery-Aware Fairness Index (RAFI) further provides a more comprehensive assessment of fairness by considering recovery opportunities for interrupted workloads. Overall, the experimental results demonstrate that APHRQ provides a balanced and efficient scheduling strategy for dynamic Hadoop YARN environments by improving responsiveness without sacrificing execution stability or fairness.

Future work will focus on implementing the proposed APHRQ framework in a real Hadoop YARN cluster to validate its effectiveness under large-scale distributed environments and heterogeneous workloads. The framework can also be extended by incorporating adaptive checkpoint placement, energy-aware resource allocation, container locality optimization, and fault-tolerant scheduling strategies to further improve scheduling efficiency and cluster performance. In addition, evaluating APHRQ on cloud-native big data platforms and in multi-cluster environments will provide further insight into its applicability to next-generation distributed computing systems.

Declarations

• **Conflict of Interest:** The authors declare that there is no conflict of interest regarding the publication of this paper.
• **Funding:** The authors received no financial support or funding for the research, authorship, or publication of this article.
• **Acknowledgements:** The authors have no acknowledgements to declare.
• **Availability of Data and Materials:** The data and materials generated or analyzed during this study are not publicly available, as they are being retained for further research. However, they may be made available by the corresponding author upon reasonable request.
• **Use of Artificial Intelligence Chatbots:** During the preparation of this manuscript, Grammarly was used for grammatical and language corrections, QuillBot was used to assist with the organization and formatting of references, and ChatGPT was used to support language refinement and the organization of the literature review. All AI-assisted content was critically reviewed, verified, and revised by the authors. These tools were not used to generate the experimental data, perform the analysis, interpret the results, or formulate the conclusions. The authors take full responsibility for the accuracy, originality, and integrity of the manuscript.

References

1. Pandey, V., Saini, P., Gupta, M., Bhardwaj, A.: “Dynamic job scheduling for energy savings in YARN: A Comparative Study with Benchmark Workloads”; Concurrency and Computation Practice and Experience, 37, 15–17 (2025). https://doi.org/10.1002/cpe.70144
2. Manjaly, J. S., Subbulakshmi, T.: “Performance Improvement through Novel Adaptive Node and Container Aware Scheduler with Resource Availability Control in Hadoop YARN”; Computer Systems Science and Engineering, 47, 3 (2023), 3083–3108. https://doi.org/10.32604/csse.2023.036320.
3. Liao, X., Zhang, H., Koshimura, M., Huang, R., Li, F.: “Solving Restricted Preemptive Scheduling on Parallel Machines with SAT and PMS”; JUCS - Journal of Universal Computer Science, 29, 8 (2023), 911–937. https://doi.org/10.3897/jucs.97743.
4. Ahmed, S. S., Slimani, Y., Frefita, R.: “Fault tolerance model for Hadoop Distributed System”; JUCS - Journal of Universal Computer Science, 31, 1 (2025), 72–92. https://doi.org/10.3897/jucs.120840.

5. Shabestari, F., Rahmani, A. M., Navimipour, N. J., Jabbehdari, S.: "A YARN-based Energy-Aware Scheduling Method for Big Data Applications under Deadline Constraints"; Journal of Grid Computing, 20, 4 (2022). https://doi.org/10.1007/s10723-022-09627-w.
6. Abraham, O. L., Ngadi, M. A. B., Sharif, J. B. M., Sidik, M. K. M.: "Task Scheduling in Cloud Environment–Techniques, Applications, and Tools: A systematic literature review"; IEEE Access, 12 (2024), 138252–138279. https://doi.org/10.1109/access.2024.3466529.
7. Nandagopal, M., Manavalan, T., Kumar, K. P., Manogaran, N., Kesavan, D., Kumar, G., Al-Khasawneh, M. A.: "Enhancing energy efficiency in cloud computing through task scheduling with hybrid cuckoo search and transformer models"; Discover Computing, 28, 1 (2025). https://doi.org/10.1007/s10791-025-09716-w.
8. Pei, H., Gu, Y., Sun, Y., Wang, Q., Liu, C., Chen, X., Cheng, L.: "LLM-based cost-aware task scheduling for cloud computing systems"; Journal of Cloud Computing Advances Systems and Applications, 14, 1 (2025). https://doi.org/10.1186/s13677-025-00822-0.
9. Wang, Y., Dong, S., Fan, W.: "Task scheduling mechanism based on reinforcement learning in cloud computing"; Mathematics, 11, 15 (2023), 3364. https://doi.org/10.3390/math11153364.
10. Raju, L. R., Reddy, M. V. K., Surukanti, S. R., Sudhakar, G., M, V. V. S. S., Adepu, A.: "IntelliScheduler: an edge-cloud computing environment hybrid deep learning framework for task scheduling based on learning"; Scientific Reports, 16, 1 (2026b). https://doi.org/10.1038/s41598-026-41330-8.
11. Yamsani, N., P, C. R.: "SLA aware deep reinforcement learning for adaptive EdgeCloud task scheduling"; Scientific Reports, 16, 1 (2026). https://doi.org/10.1038/s41598-026-40237-8.
12. Sravan, P., Shaik, M. A.: "DRL-based multi-objective task scheduling for edge-cloud computing: latency, energy, and SLA optimisation"; Scientific Reports (2026). https://doi.org/10.1038/s41598-026-49824-1.
13. Sefati, S. S., Keymasi, M., Craciunescu, R., Maiduc, S., Bayram, M., Arasteh, B.: "Adaptive resource scheduling in Multi-Cloud computing using recurrent neural forecasting and Memory-Based metaheuristic optimization"; Journal of Grid Computing, 23, 4 (2025). https://doi.org/10.1007/s10723-025-09812-7.
14. Alejandro, R. A., Helio, Z. C. P., Ángel, P. C., Francisco, V. V. J.: "A privacy-preserving efficient location-sharing scheme for mobile online social network applications."; Ulster University Research Portal (Ulster University) (2018). https://doi.org/10.1109/access.2017.doi.
15. Khaleel, M. I., Safran, M., Alfarhood, S., Zhu, M.: "Workflow Scheduling Scheme for optimized reliability and End-to-End delay control in cloud Computing using AI-Based modeling"; Mathematics, 11, 20 (2023), 4334. https://doi.org/10.3390/math11204334.
16. Gurrala, R. R., Tallapally, S. K.: "QUANTUM based latency and cost aware task scheduling in a Multi-Cloud environment"; Journal of Grid Computing, 24, 1 (2026). https://doi.org/10.1007/s10723-026-09825-w.
17. Yan, L.: "A novel quality of service-aware task scheduling approach for cloud computing using modified Wombat Optimization Algorithm"; Journal of Engineering and Applied Science, 72, 1 (2025). https://doi.org/10.1186/s44147-025-00628-6.
18. Hosseinlou, F., Ghaffari, A., Mirzaei, A., Kazem, A. A. P. H.: "Energy-aware priority-based task scheduling in cloud data centers using bacterial foraging optimization"; Scientific Reports (2026). https://doi.org/10.1038/s41598-026-50060-w.
19. Abdel-Basset, M., Mohamed, R., Elkhalik, W. A., Sharawi, M., Sallam, K. M.: "Task scheduling approach in cloud computing environment using hybrid differential evolution"; Mathematics, 10, 21 (2022), 4049. https://doi.org/10.3390/math10214049.
20. Mikram, H., Kafhali, S. E.: "CHPSO: an efficient algorithm for task scheduling and optimizing resource utilization in the cloud environment"; Journal of Grid Computing, 23, 2 (2025). https://doi.org/10.1007/s10723-025-09803-8
21. Abraham, O. L., Ngadi, M. A., Sharif, J. B. M., Sidik, M. K. M., Kolawole, O. T.: "Task scheduling in cloud computing environment based on Dwarf Mongoose optimization"; International Journal of Advanced Computer Science and Applications, 16, 12 (2025). https://doi.org/10.14569/ijacsa.2025.0161273.
22. Ahmad, W., Silaparasetti, L., Arya, P., Pandya, S., Kumar, P., Khan, T., Alabdulatif, A.: "Energy-constrained workflow scheduling for DVFS-enabled heterogeneous systems in cloud computing environments"; Scientific Reports (2026). https://doi.org/10.1038/s41598-026-49785-5.
23. Hamid, L., Jadoon, A., Asghar, H.: "Comparative analysis of task level heuristic scheduling algorithms in cloud computing"; The Journal of Supercomputing, 78, 11 (2022), 12931–12949. https://doi.org/10.1007/s11227-022-04382-x.
24. Ahmed, A., Adnan, M., Abdullah, S., Ahmad, I., Alturki, N., Jamel, L.: "An Efficient task scheduling for cloud computing platforms using energy Management algorithm: A comparative analysis of workflow execution time"; IEEE Access, 12 (2024), 34208–34221. https://doi.org/10.1109/access.2024.3371693.

25. Qureshi, B.: "Optimizing hadoop scheduling in Single-Board-Computer-Based heterogeneous clusters"; Computation, 12, 5 (2024), 96. https://doi.org/10.3390/computation12050096.
26. G, N. K., C, G. K.: "User Task Priority Based Resource Allocation with Multi Class Task Scheduling Strategy and Load Balancing in Cloud Environment"; SN Computer Science, 5, 8 (2024). https://doi.org/10.1007/s42979-024-03290-6.
27. Bal, P. K., Mohapatra, S. K., Das, T. K., Srinivasan, K., Hu, Y.-C.: "A Joint Resource Allocation, Security with Efficient Task Scheduling in Cloud Computing Using Hybrid Machine Learning Techniques"; Sensors, 22, 3 (2022b), 1242. https://doi.org/10.3390/s22031242
28. Zhou, H.: "A novel approach to cloud resource management: hybrid machine learning and task scheduling"; Journal of Grid Computing, 21, 4 (2023). https://doi.org/10.1007/s10723-023-09702-w.
29. Mindil, A., Hamed, A. Y., Hassan, M. R., Elnahary, M. Kh.: "A novel approach for dynamic task scheduling for IOT in fog-cloud environment"; Scientific Reports, 16, 1 (2026), 5501. https://doi.org/10.1038/s41598-026-35156-7.
30. V, S., M, P., P, M. K.: "Energy-Efficient task scheduling and resource allocation for improving the performance of a Cloud–FOG environment"; Symmetry, 14, 11 (2022), 2340. https://doi.org/10.3390/ sym14112340.
31. Zhang, X.: "A fine-grained task scheduling mechanism for digital economy services based on intelligent edge and cloud computing"; Journal of Cloud Computing Advances Systems and Applications, 12, 1 (2023). https://doi.org/10.1186/s13677-023-00402-0.
32. Stanisic, S., Djordjevic, B., Belotic, B., Ristic, O., Tot, I., Zivanovic, K., Kolasinac, D.: "Latency-Aware orchestration of microservices in heterogeneous kubernetes clusters using reinforcement learning"; JUCS - Journal of Universal Computer Science, 32, 4 (2026), 555–583. https://doi.org/10.3897/jucs.166567.