
Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
 Volume 25, Issues 2, pp. 1235–1246, DOI 10.12694/scpe.v25i2.2655

OPTIMAL USAGE OF RESOURCES THROUGH QUALITY AWARE SCHEDULING IN CONTAINERS BASED CLOUD COMPUTING ENVIRONMENT

S.A. POOJITHA*AND K. RAVINDRANATH†

Abstract. For cloud computing, the Quality Aware Scheduling of Containers (QASC) model has been proposed for delay-sensitive tasks. Plan your cloud tasks. Typically, time constraints are met while resources are used effectively. It's a really difficult undertaking. In order to distribute containers more effectively, QASC takes into account a number of performance factors. Containers and their make-span logs, as well as input quality metrics like I/O-intensive workload, startup time, hot standby failure rate, and inter-container dependencies, are collected by the QASC model. A metric coefficient that indicates each container's overall rating is calculated by normalizing and averaging these values to determine it. In order to determine how well scheduling performed, the model also includes a quality coefficient that calculates this metric-coefficient threshold. It's also critical for QASC to be able to determine the remaining energy in each container, which represents its request capacity. In order to optimize cloud resources, energy use is also taken into account by the model. From the cloud-sim simulation, an experimental dataset including 50 containers and 1,200 internet protocol-capable users was employed. For the make-span ratio, round-trip time, and energy consumption analysis, this produced 20,000 data points. The RLSched, DSTS, and ADATSA models were contrasted with the QASC model. The outcomes showed that QASC performed better than these models in a number of crucial areas. Tasks may be managed better with the higher average make-span ratio and lower volatility. Its superior job scheduling and resource use were further demonstrated by its shorter round-trip durations and lower energy usage across loads. The QASC model is an extremely complex scheduling method for container-based systems and a significant advancement in cloud computing research. Its approaches and methods enable for more intelligent energy use as well as high-quality services while also improving system performance, particularly for tasks that are delay-sensitive.

Key words: Cloud Computing, Resource Scheduling, Task Scheduling, Container Scheduling, Simple Moving Averages

1. Introduction.

Introduction. The cloud computing heavily relies on resource sharing to deliver services, the information technology industry has undergone a revolution. Businesses can operate more successfully because they can scale services as necessary and only pay for the resources they actually use thanks to the inherent flexibility of cloud environments [1]. However, managing resources in such a dynamic environment makes scheduling tasks in a containerized setup particularly challenging.

Containers have quickly become the go-to method for deploying applications in cloud environments thanks to their lightweight design and ability to package an application with all of its dependencies. However, as the number and variety of containerized applications increase, the task of efficiently scheduling these containers on a limited set of resources gets harder. Effectively managing this diversity while taking into account each container's unique characteristics, such as the type and intensity of the workload, the startup time, the failure rate, and the dependencies between different containers, is challenging [2].

The main focus of scheduling algorithms used in these environments has typically been on optimizing resource usage based on fundamental metrics, such as CPU or memory usage [3]. Nevertheless, these metrics frequently fall short of fully encapsulating the operational complexity of the container, leading to inefficient resource management and subpar system performance. A more sophisticated approach to container scheduling is therefore urgently needed, one that considers a wider range of metrics that reflect the complexity of containerized workloads.

We present the Quality Aware Scheduling of Containers (QASC), a novel model for container scheduling, to meet this need. The fundamental tenet of the QASC model is to give equal weight to both resource optimization and the quality of the services offered by containerized applications. This quality-conscious approach

*Department of CSE, Koneru Lakshmaiah Education Foundation, Green Fields, Vaddeswaram, Andhra Pradesh, India (Corresponding author, pranitha540@gmail.com).

†Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Andhra Pradesh, 522501, India (ravindra_ist@kluniversity.in).

is especially pertinent in the modern digital environment, where service level agreements and user experience are key factors in how valuable people perceive a service. In our study, we designed the QASC model to take into account a wide range of metrics, including metrics for I/O-intensive workload, container startup time, container failure rate, and inter-container dependency. We believe that the QASC model offers a more balanced and efficient method of managing resources in a cloud computing environment by incorporating these various metrics into the scheduling decision-making process. The QASC model's design, application, and performance assessment are discussed in this paper. We discuss the justification for each of the individual metrics included in the model and give a thorough description of each one. We then go over how these metrics are applied to scheduling decisions and offer information on how this strategy results in better resource utilization and higher-quality services. The QASC model performs significantly better than conventional scheduling algorithms in terms of resource utilization rates, system throughput, and service quality, according to our preliminary performance evaluations. These encouraging findings imply that our quality-conscious method of container scheduling presents an important new tool for resource management in cloud computing environments.

This paper advances the ongoing discussion on effective resource management in cloud computing environments and provides new opportunities for container scheduling optimization. We hope that our work will encourage additional investigation into this crucial area, resulting in even more advanced and efficient solutions in the future.

The majority of container scheduling tools rely on heuristics, such as different packing methods using heuristics, such Best-Fit Decreasing (BFD) as Well as first Decreasing (FFD). The Google Kubernetes default scheduler [4] provides a number of heuristics, such as affinities or load level, which may be used to get the desired result. The random and binpack policies that Docker Swarm offered in the past gave priority to PMs having highest resource use. The contemporary version [5] simply provides the Spread algorithm, which aims to distribute the workload among PMs. Spread is expanded by Lu Ni et al. [6] using a linear weighted approach that takes into account other characteristics, such the amount of free RAM. Heuristics can work rather well in general for the particular aim they are intended to achieve, but they lack flexibility.

Utilizing numerous optimization strategies is a popular tactic in the state-of-the-art. In order to solve the multi-objective optimization issue of work scheduling in containers, Zhang et al. [7] suggest a linear programming approach. As one might anticipate, the method fails to recognize the non-linearities associated with the patterns of resource utilisation that a deep neural network can (with activation functions). Genetic algorithms and other evolutionary optimization techniques are widely used in publications. In order to balance resource utilisation, Kaewkasi et al., [8] present a container scheduling method based on the ant colony optimization algorithm. Genetic algorithms are used by Zhang et al., [9] to increase the data center's total energy efficiency. To achieve this aim of boosting the count of clients serviced, Mseddi et al., [10] use a metaheuristic that utilises PSO (particle swarm optimization) for container placement as well as work scheduling in fog settings. However, all of the investigated optimization techniques base their placement choices purely on the data center's existing condition. It is feasible to construct an ideal policy by learning the environmental characteristics (task arrival pattern, resource utilisation in PMs), as demonstrated by contemporary approaches. The potential to adapt past scenarios of the contextual dynamics emerged during container scheduling, which has not yet been fully investigated.

Since the resource management is the difficult in cloud environments, the explored contemporary methods being successfully applied to a number of resource management issues there, including auto-scaling [11], placement issues [12] of virtual machines, as well as container scalability [13]. The container scalability assessment approach SARSA (a Q-learning variation) is suggested by Zhang et al. [13] as a method for managing the horizontal flexibility and scalability of containers. Zhiguang Wang et al., [11] presented a DQN and D3QN-based VM auto-scaling solution. Qingchen Zhang et al., [14] suggested an approach to handle tasks using in edge settings. Experiments will demonstrate that value-iteration techniques, however, may converge to ineffective policies. As an alternative, PPO, a policy-based technique, is suggested for use in Funika et al., [15] proposed DDQN for application auto-scaling.

A "hybrid optimal and deep learning technique for dynamic scalable task scheduling (DSTS) in container cloud environment" was presented by Saravanan M et al., [16]. They proposed a modified multi-swarm coyote optimization (MMCO) approach to increase containers' virtual resources, which enhances client service level

agreements. The assurance of priority-based scheduling follows. In addition, a "fast adaptive feedback recurrent neural network (FARNN)" for pre-virtual CPU allocation and a modified pigeon-inspired optimization (MPIO) approach for task clustering were developed. A "deep convolutional neural network (DCNN)" is used to power the task load monitoring system, which is designed to provide dynamic priority-based scheduling.

A "self-adapting task scheduling algorithm" in short titled as ADATSA utilising learning automata was suggested by Lilu Zhu et al.,[17] to solve environmental dynamics such complicated dependence relationships and quick iterations of container scheduling. They developed a useful reward-penalty method for scheduling activities in conjunction with the present environment's running tasks as well as idle resource states.

A "self-adaptive deep reinforcement learning-based scheduler (RLSched)" that automatically captures the resource utilisation dynamics in the data centre has been developed by T. L. Botran et al., [18]. A decentralised actor-critic multi-agent architecture, which allows for parallel execution and quicker convergence, served as the foundation for this scheduler. RLSched is based on an improved network architecture including action shaping that filters out improper activities and keeps the agent from adopting a poor policy.

However, most of the contemporary approaches endeavoured to maximize the performance of the container utilization, they are centric to resource management problems but not considering the contextual dynamics raised during the containers scheduling. Hence, it is obvious to encourage the present research towards container scheduling with regard to contextual dynamics such as roundtrip time, energy consumed and available, as well as computational-load processing time.

To the best of our knowledge, the suggested technique QASC is the first effort to investigate the efficacy of contextual dynamics of the containers using a simple-moving-average that is extremely scalable. QASC is centred to the contextual dynamics of the containers, as opposed to other ideas, since it focuses on optimising the usage of the containers serviced and hence increases predicted performance.

2. Methods and Materials. In the context of scheduling containers to delay-sensitive tasks, the Quality Aware Scheduling of Containers (QASC) model assumes a significant role in ensuring task completion within the required time frames while optimizing resource usage. QASC is a method for streamlining the process of allocating jobs to available containers in a computational system as shown in figure 2.1. The main objective of QASC is to increase scheduling efficiency while taking into account a variety of variables that may have an impact on system performance as a whole. This system accepts a set of containers as input, along with their corresponding logs of makespans (the total amount of time required to complete a schedule) and specific metrics for each container, such as I/O-intensive workload, startup time, failure rate, and inter-container dependencies. QASC establishes initial, minimum, maximum, and end values in a normalized form for each container and each metric. In order to track trends over time, it then computes the simple moving average of each of these progression measures. Further exploring these moving averages' patterns, QASC uses them to determine a metric coefficient for each measure. This coefficient offers a thorough evaluation of a container's performance by combining the mean and root mean square distance of progression measures. Additionally, the system determines a quality coefficient for every container that provides an estimate of the metric-coefficients threshold. This threshold essentially assesses the effectiveness of scheduling while taking into account the unique demands of each container and its characteristics. The figure 2.1 outlines the process of the Quality Aware Scheduling of Containers (QASC) system. Here's a detailed explanation:

User Input: The user provides the QASC system with a set of containers (CT) and their corresponding logs of make-spans, as well as specific metrics for each container. These metrics include:

I/O Intensive Workload (ioiw)

Container Startup Time (cst)

Container Failure Rate (cfr)

Inter-container Dependencies (icd) **Normalized Progression Measure Calculation**: For each container and its respective metrics, the system calculates normalized progression measures ($x_{(}norm)_{ct}$).

Simple Moving Average (SMA) Calculation: The system computes the simple moving average of each of the progression measures for every metric.

Progress Measurement Patterns: It then discovers patterns in the progression measurement.

Metric Coefficient Calculation: After identifying patterns, the system calculates the Metric Coefficient for each metric. This involves computing the mean () and the root mean square distance (RMSD) of progression

measures. The metric coefficient (mc) for each metric is computed as the sum of its mean and RMSD.

Quality Coefficient Calculation: The system calculates the Quality Coefficient for each container.

Metric-Coefficients Threshold Estimation: The system estimates the metric-coefficients threshold (qc_ct) for each container as a function of the metric coefficients and specific metrics.

Residual Energy Estimation: For each container, the system estimates the residual energy (RE_{ct}), which is the energy left after accounting for the energy consumption (E_{ct}) and the product of energy coefficient (EC_w) and the container-specific weight (w_{ct}).

Container Ranking: The containers are then ranked in descending order based on their metric coefficient thresholds (qc_{ct}).

Container Selection: Based on the ranking, the system selects the optimal container for scheduling.

Optimal Scheduling: Finally, the optimal scheduling of containers is performed, completing the QASC process.

The capability of QASC to calculate each container's residual energy, which represents the container's ability to fulfill the request, is another important feature. Additionally, the system figures out energy consumption, which is the difference between the starting point and the anticipated residual value.

The best container for scheduling is then chosen after containers are ranked according to their metric coefficient thresholds. Thus, the QASC approach considers a number of variables to optimize container scheduling, with the goal of enhancing system performance as a whole.

For analysis and predictive scheduling, the QASC model computes Simple Moving Averages (SMAs) based on beginning value, maximum value, and minimum representational form. SMA allows for the following: beginning times, failure rates, determining basic I/O burden, smoothing volatility, and dependency on other containers. Understanding how containers function over long distances and minimizing the effects due to transient anomalies make smoothing important. These SMA values may be used by QASC to more precisely predict the susceptibility of containers. As a result, job distribution might be based on the functional profile of each container from inception to completion, baseline, and peak. So that no container is under scheduled due to a conservative policy or overloaded when demand increases, these predictive insights are important in resource allocation. This significantly raises the reliability and efficiency of cloud computing.

Here are the QASC factors in the context of delay-sensitive tasks:

- I/O **Intensive Workload:** The I/O operations handled by a container during the average makespan time for delay-sensitive tasks are considered. Containers with higher I/O workloads might experience higher latency, which could potentially affect delay-sensitive tasks.
- **Container Startup Time:** The startup time of a container is a critical factor when dealing with delay-sensitive tasks. A container with a shorter startup time can begin processing tasks faster, reducing the risk of delays.
- **Container Failure Rate:** A container with a higher failure rate may disrupt the timely completion of tasks. This is especially crucial for delay-sensitive tasks, where any failure can lead to unacceptable delays.
- **Inter-container Dependencies:** The dependencies between containers can also impact the scheduling of delay-sensitive tasks. A container with fewer dependencies is less likely to be delayed due to issues with other containers, which makes it more suitable for delay-sensitive tasks.

The Quality Score (QS) calculated based on these metrics guides the QASC model in allocating tasks to containers. Containers with higher QS, indicating lower I/O workload, faster startup time, lower failure rate, and fewer dependencies, are prioritized for delay-sensitive tasks to ensure timely completion and optimize the usage of resources.

2.1. Quality Aware Scheduling of Containers (QASC). Input: Set of containers CT, their corresponding log of makespans, container specific metrics: (1). I/O Intensive Workload ($ioiw$), (2). Container Startup Time (cst), (3). Container Failure Rate (cfr), and (4). Inter-container Dependencies (icd). **Output:** Optimal scheduling of containers. **Initialization:** For each container $ct \in CT$, For each makespan $m \in log$ that successfully completed, For each metric $i \in ioiw, cst, cfr, icd$, Determine the initial (o), minimum (l), maximum (h), and end (e) value of the metric in normalized form as follows. For each metric $i \in ioiw, cst, cfr, icd$ and for each container $ct \in CT$, Calculate the normalized progression measure x_{norm} for initial (o), min (l), max

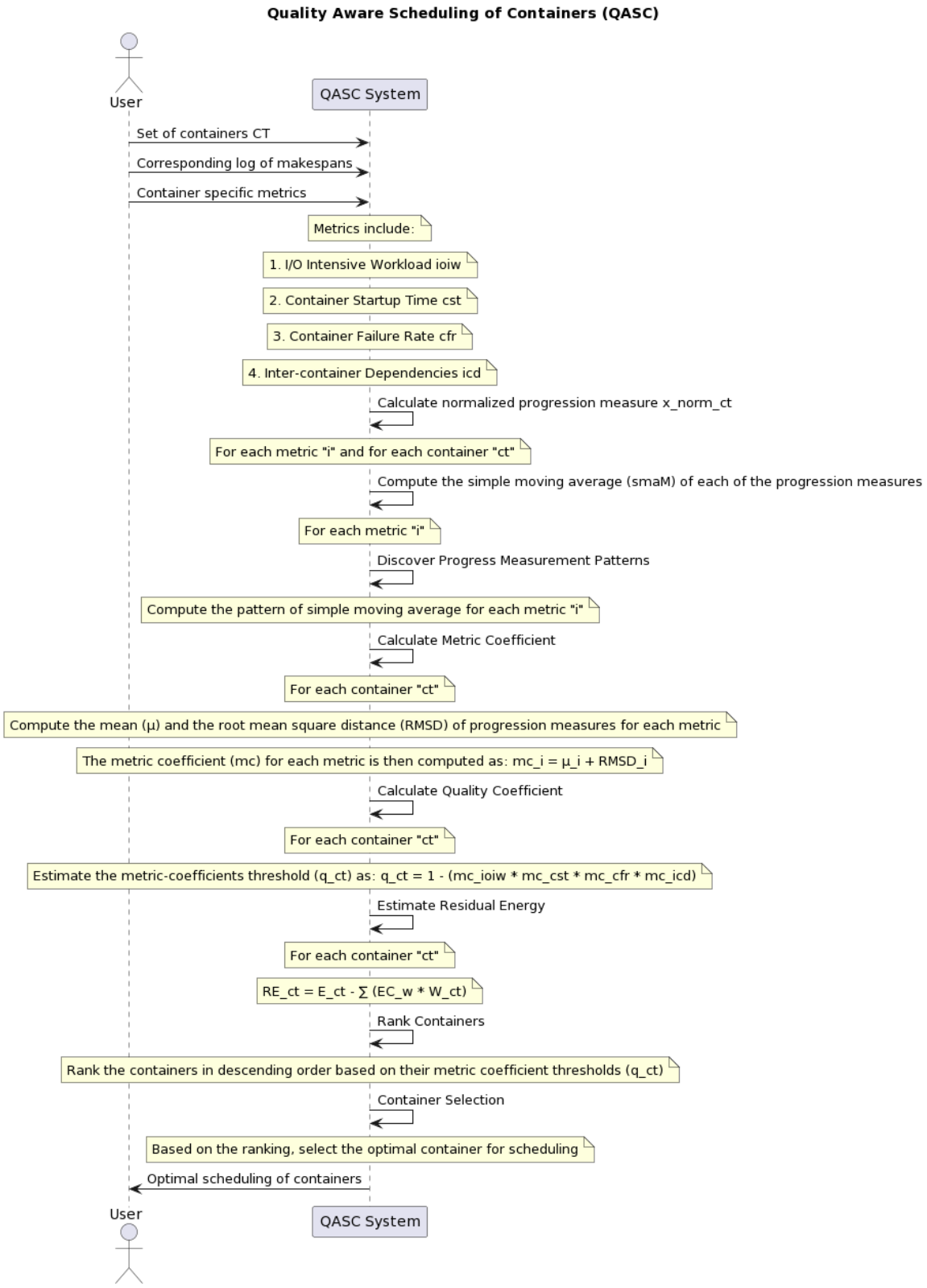

Fig. 2.1: The flow diagram representation of the QASC

(h), and end (e) values using the formula: Eq 2.1

$$x_{(norm_{ct})} = 1 - (1/x_{ct}) \tag{2.1}$$

where x_{ct} is the original progression measure for the metric i in the container ct, and $x_{norm_{ct}}$ is the normalized progression measure. Compute the simple moving average of each of the progression measures $(o,\ l,\ h,\ e)$, for each metric i as $smaM_i$ follows: For a set of metrics $M = m_1, m_2, ..., m_n$ and a simple moving average configuration $smac$, the simple moving average of the i^{th} metric's initial values, maximum values, minimum values, and end values can be represented by four equations:"

- Simple moving average of initial values $(sma_i(o))$: $sma_i(o) = (1/smac) * \sum_{j=i}^{i+smac} m_j(o)$
- Simple moving average of maximum values $(sma_i(h))$: $sma_i(h) = (1/smac) * \sum_{j=i}^{i+smac} m_j(h)$
- Simple moving average of minimum values $(sma_i(l))$: $sma_i(l) = (1/smac) * \sum_{j=i}^{i+smac} m_j(l)$
- Simple moving average of end values $(sma_i(e))$: $sma_i(e) = (1/smac) * \sum_{j=i}^{i+smac} m_j(e)$

where:

- $m_j(o)$, $m_j(h)$, $m_j(l)$, $m_j(e)$ represent the initial value, maximum value, minimum value, and end value of the j^{th} metric respectively.
- The summation $\sum_{j=i}^{i+smac}$ indicates that the sum is taken from the i^{th} metric to the $(i+smac)^{th}$ metric.

Finally, the set of these simple moving averages $smaM$ for all metrics i in M is given by: $smaM = sma_i(o)$, $sma_i(h), sma_i(l), sma_i(e)$ for all i in M

Discover Progress Measurement Patterns:

For each metric $i \in ioiw, cst, cfr, icd$,

Compute the pattern of simple moving average for each progression measure as:

- $psmaM_i(o) = smaM_i(o)$ if $smaM_i(o)$ belongs to the first makespan
- Else,
- $psmaM_i(o) = (psmaM_{(i-1)}(o) + psmaM_{(i-1)}(e))/2$
- $psmaM(e) = (smaM(o) + smaM(h) + smaM(l) + smaM(e))/4$
- $psmaM(l) = \min(smaM(l), psmaM(o), psmaM(e))$
- $psmaM(h) = \max(smaM(h), psmaM(o), psmaM(e))$

Calculate Metric Coefficient: For each container $ct \in CT$, Compute the mean (μ) and the root mean square distance (RMSD) of progression measures for each metric as

$$mu_i = (\text{psmaM}_i(o) + \text{psmaM}_i(h) + \text{psmaM}_i(l) + \text{psmaM}_i(e)/4) \tag{2.2}$$

$$RMSD_i = \sqrt{((\text{psmaM}_i(o) - \mu_i)^2 + (\text{psmaM}_i(h) - \mu_i)^2 + (\text{psmaM}_i(l) - \mu_i)^2 + (\text{psmaM}_i(e) - \mu_i)^2)/4} \tag{2.3}$$

The metric coefficient (mc) for each metric is then computed as

$$mc_i = \mu_i + \text{RMSD}_i \tag{2.4}$$

Calculate Quality Coefficient: For each container $ct \in CT$, Estimate the metric-coefficients threshold (q_{ct}) as

$$q_{ct} = 1 - (mc_{ioiw} * mc_{cst} * mc_{cfr} * mc_{icd}) \tag{2.5}$$

Estimate Residual Energy: For each container $ct \in CT$, Let's denote:

E_{ct} as the current energy level of container ct W_{ct} as the workload of container ct EC_w as the estimated energy consumption of workload w Then, the Residual Energy (RE) of a container ct can be calculated as follows

$$RE_{ct} = E_{ct} - \sum(EC_w * W_{ct}) \tag{2.6}$$

Estimate the residual energy (re_{ct}) that indicates the container's scope to complete the request. Also, estimate the energy consumption (ec_{ct}) which is the unconditional disparity between the initial-value and the predicted residual-value.

Table 3.1: Record of Progress Measures

Container	Make span	Metric	Initial	Maximum	Minimal	Residual

Table 3.2: Mean Make-Span Ratio of the QASC, DSTS, ADATSA, and RLSched

No. of Containers	QASC	DSTS	ADATSA	RLSched
10	31	18	15	12
15	33	20	17	14
20	36	20	17	14
25	39	24	21	18
30	41	32	29	26
35	44	33	30	27
40	44	33	30	27
45	47	33	30	27
50	49	34	31	28

Rank Containers: Rank the containers in descending order based on their metric coefficient thresholds (q_{ct}).

Container Selection: Based on the ranking, select the optimal container for scheduling.

The quality of the scheduling is determined by considering factors such as the container's I/O workload, startup time, failure rate, inter-container dependencies, and energy consumption. The selection of the optimal container is done based on the calculated metric and quality coefficients, and the anticipated residual energy [19]."

3. Experimental study. The goal of the experimental investigation was to expand the efficiency of the suggested scheduling model in order to schedule containers to requests. The experiments were conducted on a dataset [20], which represents the metrics of each make-span in series observed from each container. The dataset shows a collection of records with the following table 3.1 structure. Cloud-sim based simulation has been used to create the dataset mentioned above. The dataset was created by taking into account all 1,200internet protocol-capable users. Through the scheduling gateway, this Internet of Things network is connected to the resource provider. Furthermore, a cloud computing network with 50 containers is connected to these resource providers. Each container's most recent 20 make-spans are used to compile the metric values. The dataset generated has a total of 20,000 records included in it. The average make-span-ratio, roundtrip-time against a varied count of containers, as well as adaptable transactional requests load are the parameters taken into account for performance study. Energy consumption ratio versus fluctuating load, another key metric, has also been evaluated.

The comparison of metric values acquired from the QASC with the comparable results from the existing models DSTS [16], ADATSA [17], as well as RLSched [18], denoting the efficiency of the QASC.

The amount of time needed to do the activity is indicated by the metric "make-span ratio". The make-span ratio is predicted to be proportional to the count of containers in Table 3.2, Figure 3.1. According to QASC scheduling model, the mean make-span ratio of 21.7±5.1 is being noted. In contrast to the mean make-span ratios of the DSTS (24.4±3.4), ADATSA (27.5±3.4), and RLSched model (40.4±0.4), the mean make-span ratio of the QASC is noticeably high and has a little variance.

For QASC, DSTS, ADATSA, and RLSched, another quality metric stated as roundtrip time is being taken into account and compared (see Table 3.3, Figure 3.2). Figure 3.2 statistics, which show that the average roundtrip times observed from the models QASC, DSTS, ADATSA, and RLSched are respectively 20.4±6.6, 36.5±6.4, 40.5±10.4, and 44.5±14.4, indicate that the model QASC outperforms the existing models DSTS, ADATSA, as well as RLSched with shorter roundtrip times. In Table 3.4, Figure 3.3, the roundtrip time under varied load is predicted. Comparing the performance of the model QASC to that of the existing methods DSTS,

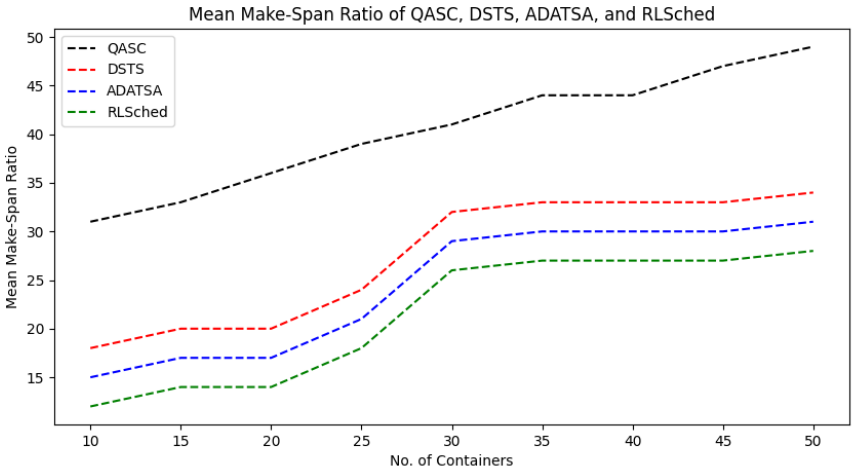

Fig. 3.1: The mean make-span ratio of the QASC, DSTS, ADATSA, and RLSched

Table 3.3: Between a Fixed Load and an Adaptable Number of Containers, the Mean Roundtrip Time

No. of Containers	QASC	DSTS	ADATSA	RLSched
10	24	45	49	53
15	29	43	47	51
20	26	43	47	51
25	25	40	44	48
30	22	36	40	44
35	20	34	38	42
40	17	32	36	40
45	13	31	35	39
50	7	25	29	33

ADATSA, as well as RLSched, it is clear that it performs better. The mean difference among the roundtrip times of the QASC (18.9±8.1) and the DSTS (34.8±7.95), the ADATSA (39.8±12.9), and the RLSched model (44.8±17.9) has been noted.

Table 3.5, Figure 3.4 shows the data of the important objective energy use. Comparisons are made between the proposed QASC and the current DSTS, ADATSA, as well as RLSched models' average energy usage (measured in joules) by each of the make-spans. The outcome demonstrates that the suggested model QASC outperforms the current models DBSA, which consumes an average of 20±11 joules per make-span, ADATSA, which consumes an average of 24±15.22 joules per make-span, and RLSched, which consumes an average of 28±19.22 joules per make-span. In comparison to the existing methods DSTS, ADATSA, and RLSched, the QASC uses less energy on average.

The improvement in the performance can be understood as a result of QASC's holistic approach, which takes into account not only the current state of the containers but also their historical performance data. This enables the QASC system to make more informed and precise scheduling decisions, leading to the observed enhancements in operational efficiency and task handling capacity.

4. Conclusion. The QASC has been put through a great deal of testing in cloud computing operations and is a substantial upgrade over existing models such as DSTS, ADATSA, and RLSched. This is demonstrated by the empirical findings of extensive experimental testing. By taking into account make-span, round-trip

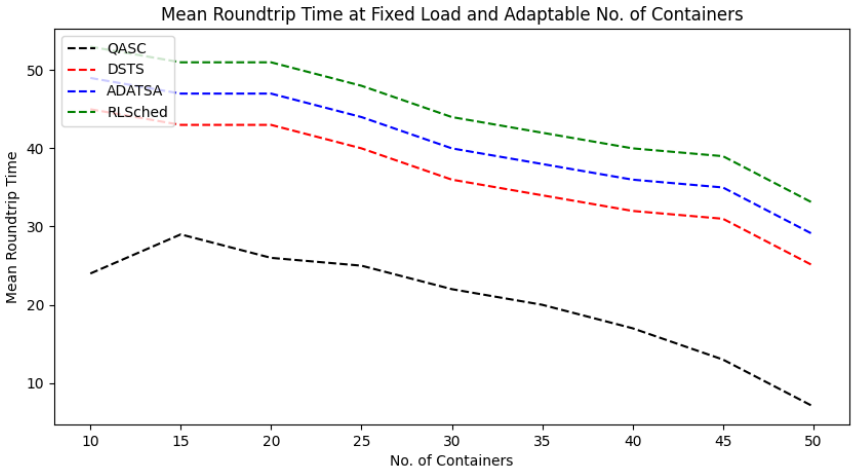

Fig. 3.2: The mean roundtrip time of QASC, DSTS, ADATSA, and RLSched observed from adaptable count of containers

Table 3.4: Mean Roundtrip Time Observed at Fixed Count of Containers vs. Adaptable Count of Requests

Adaptable Count of Requests in KBPS	QASC	DSTS	ADATSA	RLSched
10	7	24	29	34
15	10	27	32	37
20	13	30	35	40
25	18	31	36	41
30	18	31	36	41
35	19	36	41	46
40	22	42	47	52
45	29	44	49	54
50	34	49	54	59

time, energy consumption, and container properties, QASC's scheduling method enhances performance. The translation look-ahead approach lowers make-span ratios and roundtrip times when container counts change. Energy efficiency, a key component of green cloud computing, is where QASC beats existing cloud computing models. According to the energy consumption data, QASC consumes less energy for different request loads per make-span. Such efficiency is needed for large-scale cloud computing to be sustainable both financially and environmentally. Due to its efficiency in resource allocation and high quality, QASC is helpful for cloud computing. This demonstrates how balancing resource consumption and performance may greatly increase cloud efficiency. Lastly, QASC promises a bright future by enhancing cloud computing resource management performance and energy consumption. It may take the lead in cloud computing technologies in the future due to its outstanding performance and better results than those of existing models.

REFERENCES

[1] I. Ahmad, et al., *Container scheduling techniques: A survey and assessment*, Journal of King Saud University-Computer and Information Sciences, 34.7 (2022), pp. 3934-3947.

[2] K. Senjab, S. Abbas, and N. Ahmed, *A survey of Container scheduling algorithms*, Journal of Cloud Computing, 12.1 (2023), pp. 1-26.

[3] S. Singh, and I. Chana, *A survey on resource scheduling in cloud computing: Issues and challenges*, Journal of grid

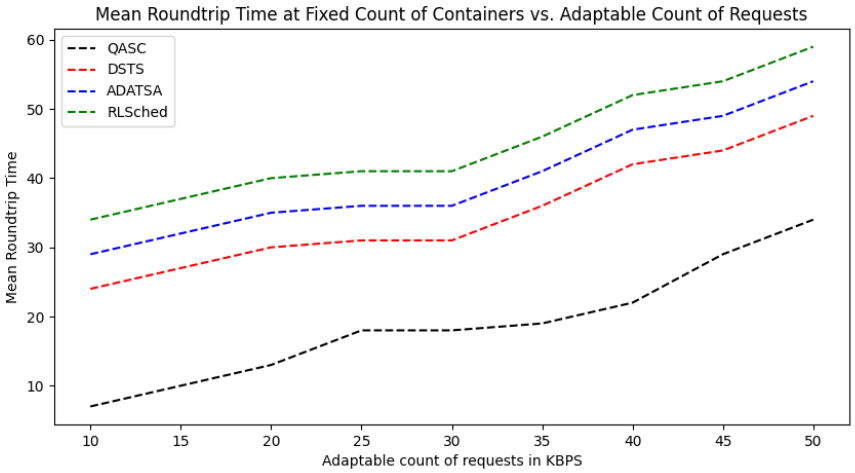

Fig. 3.3: Roundtrip time observed from the QASC, DSTS, ADATSA, and RLSched against adaptable load of requests

Table 3.5: Energy Usage (Measured in Joules) by Each of the Make-Spans at Adaptable Load of Requests

Adaptable Load of Requests in KBPS	QASC	DSTS	ADATSA	RLSched
10	1	3	7	11
15	2	8	12	16
20	7	11	15	19
25	10	17	21	25
30	14	19	23	27
35	17	22	26	30
40	19	27	31	35
45	21	34	38	42
50	25	39	43	47

computing, 14 (2016), pp. 217-264.
[4] Kubernetes, *Google Kubernetes*, URL: https://kubernetes.io/docs/concepts/schedulingeviction/kube-scheduler/ (visited on 01/2021).
[5] Docker, *Docker Swarm*, URL: https://docs.docker.com/engine/swarm/how-swarm-modeworks/services/ (visited on 01/2021).
[6] S. Lu, M. Ni, and H. Zhang, *The optimization of scheduling strategy based on the Docker swarm cluster*, Information Technology, 40.7 (2016), pp. 147–151.
[7] D. Zhang, et al., *Container oriented job scheduling using linear programming model*, In: 2017 3rd International Conference on Information Management (ICIM), IEEE, 2017, pp. 174–180.
[8] C. Kaewkasi, and K. Chuenmuneewong, *Improvement of container scheduling for docker using ant colony optimization*, In: 2017 9th international conference on knowledge and smart technology (KST), IEEE, 2017, pp. 254–259.
[9] R. Zhang, et al., *A genetic algorithm-based energy-efficient container placement strategy in CaaS*, IEEE Access, 7 (2019), pp. 121360–121373.
[10] A. Mseddi, et al., *Joint container placement and task provisioning in dynamic fog computing*, IEEE Internet of Things Journal, 6.6 (2019), pp. 10028–10040.
[11] Z. Wang, et al., *Automated cloud provisioning on aws using deep reinforcement learning*, arXiv preprint arXiv:1709.04305, 2017.
[12] B. Du, C. Wu, and Z. Huang, *Learning resource allocation and pricing for cloud profit maximization*, In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 7570–7577.
[13] S. Zhang, et al., *A-SARSA: A Predictive Container Auto-Scaling Algorithm Based on Reinforcement Learning*, 2020 IEEE International Conference on Web Services (ICWS), IEEE, 2020, pp. 489–497.
[14] Q. Zhang, et al., *A double deep Q-learning model for energy-efficient edge scheduling*, IEEE Transactions on Services

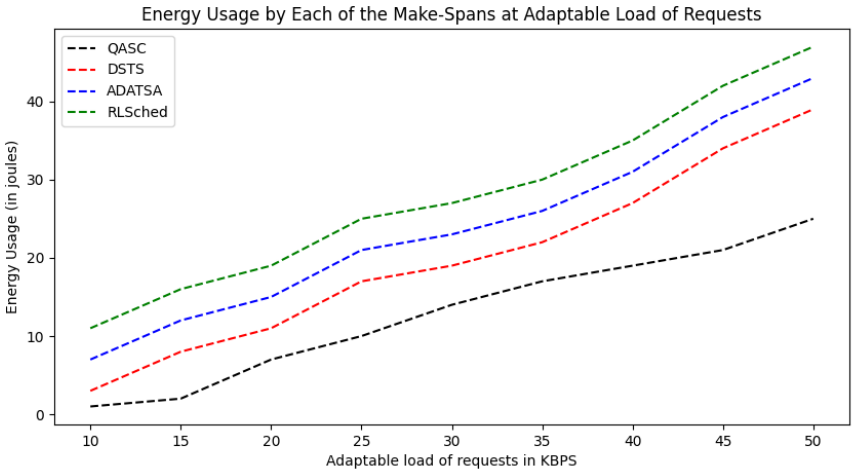

Fig. 3.4: The energy used (measured in joules) by each of the make-spans noted from the QASC, DSTS, ADATSA, and RLSched techniques

Computing, 12.5 (2018), pp. 739–749.
[15] W. Funika, J. Koperek, and P. Kitowski, *Automatic management of cloud applications with use of Proximal Policy Optimization*, International Conference on Computational Science, Springer, 2020, pp. 73–87.
[16] S. Muniswamy, and R. Vignesh, *DSTS: A hybrid optimal and deep learning for dynamic scalable task scheduling on container cloud environment*, Journal of Cloud Computing, 11.1 (2022), pp. 1–19.
[17] L. Zhu, et al., *A Self-Adapting Task Scheduling Algorithm for Container Cloud Using Learning Automata*, IEEE Access, 9 (2021), pp. 81236–81252.
[18] T. Lorido-Botran, and M. K. Bhatti, *Adaptive container scheduling in cloud data centers: a deep reinforcement learning approach*, International Conference on Advanced Information Networking and Applications, Springer, Cham, 2021.
[19] A. Gellert, A. Florea, U. Fiore, F. Palmieri, and P. Zanetti, *A study on forecasting electricity production and consumption in smart cities and factories*, International Journal of Information Management, 49 (2019), pp. 546–556.
[20] Kaggle, *Datasets*, Available at: https://www.kaggle.com/datasets.

Edited by: Anil Kumar Budati
Special issue on: Soft Computing and Artificial Intelligence for wire/wireless Human-Machine Interface
Received: Oct 4, 2023
Accepted: Jan 18, 2024