

Machine Learning-Based Workload-Aware Scheduling for IoT Data-Cube Queries in Cloud Data Centers

Mansoor Iqbal^{*1} and Muhammad Adeen Zahid²

¹Eratosthenes Center of Excellence, Limassol, Cyprus

²Department of Computer Science, University of Wah, Wah Cantt, Pakistan

September 30, 2026

Abstract

IoT applications generate multidimensional sensor data that are processed through analytical queries in cloud data centers, where variations in query complexity can cause uneven server workloads. This paper presents a machine-learning-based simulation framework for IoT data-cube workload management. The framework generates synthetic IoT data, constructs a multidimensional cube, extracts pre-execution query features (R , D , Q_r , S_c), and applies a Random Forest classifier to predict Low, Medium, and High workload classes. The classifier achieves 94.00% accuracy on the held-out test set. Predicted classes are mapped to training-derived service-demand estimates and used by a workload-aware scheduler to assign tasks based on each server's predicted remaining load. Across 20 paired simulation runs, the proposed method shows its strongest gains under High load, reducing delay by 14.64%, increasing throughput by 2.52%, improving utilization by 2.53%, and reducing load-balance dispersion by 47.59% relative to Round Robin. Under Overload, it reduces task rejection by 5.37% and delay by 1.14%. Paired tests with Holm correction confirm the significance of these principal improvements. The framework provides a reproducible basis for workload-aware IoT query scheduling in cloud data centers.

Keywords: Internet of Things; Data Cube; Workload Prediction; Workload-Aware Scheduling; Cloud Data Centers;

1 Introduction

Internet of Things (IoT) applications continuously generate sensor data in smart cities, industrial systems, and environmental monitoring [1, 2]. Cloud data centers process and analyze this data through analytical queries [3, 4], where increasing sensor density creates higher computational demand and uneven resource utilization.

Multidimensional data cubes organize IoT data across dimensions such as time, location, sensor type, and event category [5]. However, query complexity varies significantly—a query spanning one location and sensor type requires minimal processing, whereas one covering multiple locations, long time ranges, and several sensor types imposes substantial server load. Cloud schedulers must therefore consider expected task demand when assigning workloads.

^{*}Corresponding author: mansoor.iqbal@eratosthenes.org.cy

Existing approaches have been developed on the individual aspects of IoT data management, cloud scheduling, and workload prediction [6, 7, 8]. Random assignment and Round Robin are common approaches that neither account for task workload variation nor consider query characteristics. While simulation frameworks are good at modeling infrastructure, they do not support any integrated query-level workload classification or workload-aware scheduling [9, 10].

This work introduces an easy-to-use Python-based simulation framework that combines synthetic IoT data generation, multidimensional data-cube modeling, query-level feature extraction (R, D, Q_r, S_c), workload classification via Random Forest, and workload-aware server assignment into a comprehensive, reproducible pipeline.

The main contributions are:

- A reproducible synthetic IoT data-generation and multidimensional data-cube pipeline organized by time, location, sensor, and event dimensions.
- A pre-execution query workload model using estimated output cardinality (R), scanned data volume (D), query range (Q_r), and sensor coverage (S_c).
- A Random Forest classifier that predicts Low, Medium, and High workload classes and maps them to training-derived service-demand estimates.
- A workload-aware scheduling policy that assigns tasks using predicted service demand and the current predicted remaining load of each server.
- A paired evaluation against Random and Round Robin across Low, Medium, High, and Overload scenarios using 95% confidence intervals and Holm-corrected significance testing.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the proposed methodology. Section 4 describes the experimental setup. Section ?? discusses results, and Section 6 concludes the paper with future directions.

2 Related Work

2.1 Cloud and Edge Simulation Frameworks

CloudSim established a modular platform for modeling data centers, virtual machines, and scheduling policies [6]. Subsequent frameworks extended these capabilities toward IoT and edge: IOTSim supports IoT big-data and MapReduce processing [11]; iFogSim evaluates application placement across cloud, fog, and edge using latency, energy, and cost metrics [9]; IoTSim-Edge models heterogeneous IoT/edge resources with mobility and battery behavior [10]; IoTSim-Osmosis addresses workload migration across edge-cloud environments [12]; and iFogSim2 extends fog simulation with mobility, clustering, and microservices [13]. These platforms provide extensive infrastructure modeling but lack integrated support for data-cube query modeling, query-level workload prediction, and workload-aware server assignment.

2.2 Workload-Aware Scheduling and Prediction

Task scheduling directly affects execution time, utilization, and workload distribution. Classical heuristics map tasks using fixed rules [7], while cloud methods balance performance and energy under dynamic workloads [14]. However, conventional schedulers do not consider the computational complexity of multidimensional analytical queries.

Machine learning enables predictive resource management by estimating workload before allocation [8]. Decision Trees [15], Random Forests [16], Support Vector Machines [17], and

Table 1: Integrated Comparison of Simulation, Workload-Prediction, and Scheduling Approaches

Approach	Infrastructure Support			Workload Modeling and Scheduling				Evaluation and Reproducibility	
	Cloud	IoT	Edge/ Fog	Data-Cube Features	ML Prediction	Workload Assignment	Unified Pipeline	Evaluation Scope	Repro- ducible
(lr)2-4 (lr)5-8 (lr)9-10									
CloudSim [6]	✓	×	×	×	×	×	×	Resource metrics	–
Scheduling [7]	✓	×	×	×	×	Δ (static)	×	Makespan	×
ML Prediction [8]	✓	×	×	×	✓ (system)	×	×	Accuracy, statistics	×
iFogSim [9]	✓	✓	✓	×	×	×	×	Latency, energy, cost	–
IoTSim-Edge [10]	✓	✓	✓	×	×	×	×	Infrastructure	–
IOTSim [11]	✓	✓	×	Δ	×	×	Δ	Latency, cost	✓
IoTSim-Osmosis [12]	✓	✓	✓	×	×	×	×	Placement, migration	–
iFogSim2 [13]	✓	✓	✓	×	×	×	×	Mobility, resources	–
Proposed	✓	✓	✓	✓ (query)	✓ (RF)	✓ (load)	✓	5 metrics, stats	✓

✓ = supported; Δ = partial; × = not reported; – = not assessed.

neural networks [18] have been applied to workload classification. Most studies rely on system-level metrics (CPU, memory), whereas the proposed framework derives workload from query-level features (R , D , Q_r , S_c).

Table 1 summarizes the reviewed approaches across infrastructure support, workload modeling, scheduling, evaluation scope, and reproducibility.

As shown in Table 1, existing approaches address infrastructure, prediction, and scheduling separately. The proposed framework uniquely integrates data-cube query features, ML-based workload classification, and workload-aware scheduling within a unified, reproducible pipeline.

3 Proposed Methodology

This study proposes a machine-learning-based simulation framework for managing heterogeneous IoT data-cube workloads in cloud data centers. As illustrated in Fig. 1, the framework integrates IoT data generation, multidimensional cube construction, pre-execution query characterization, workload prediction, and workload-aware server assignment. The framework predicts the expected demand of an incoming analytical query before execution and uses this information to guide task placement across cloud servers.

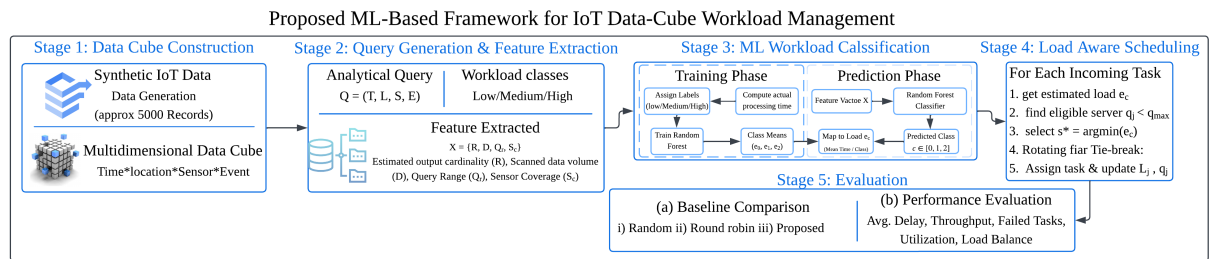


Figure 1: Proposed ML-based framework for IoT data-cube workload management, including data-cube construction, query feature extraction, workload classification, load-aware scheduling, and performance evaluation.

3.1 IoT Data-Cube and Query Modeling

The synthetic IoT dataset is represented as

$$\mathcal{D} = \{r_1, r_2, \dots, r_n\}, \quad (1)$$

where each record is defined as

$$r_i = (t_i, l_i, s_i, e_i, v_i), \quad (2)$$

with timestamp t_i , location l_i , sensor type s_i , event category e_i , and measured value v_i . The records are organized into a multidimensional data cube over time, location, sensor, and event dimensions.

An analytical query is defined as

$$Q = (T, L, S, E), \quad (3)$$

where T , L , S , and E specify the selected temporal interval, locations, sensor types, and event categories. Each query is represented before execution by the feature vector

$$X = (R, D, Q_r, S_c), \quad (4)$$

where R is the estimated output cardinality, D is the estimated scanned data volume, Q_r is the query time range, and S_c is sensor coverage. The estimates of R and D are derived from data-cube metadata and dimensional frequencies, ensuring that post-execution information is not used during workload prediction.

3.2 Machine Learning-Based Workload Prediction

A Random Forest classifier predicts the workload class

$$c \in \{0, 1, 2\}, \quad (5)$$

corresponding to Low, Medium, and High query workloads. Class boundaries are obtained from processing-time quantiles computed using the training set only. The Overload case is not an ML class; it is treated separately as a scheduling scenario in which the offered workload exceeds the available processing capacity.

The predicted class is converted into a numerical service-demand estimate using the class-mean processing time

$$e_c = \frac{1}{|\mathcal{T}_c|} \sum_{i \in \mathcal{T}_c} t_i, \quad (6)$$

where $\mathcal{T}_c$ denotes the training queries belonging to class c and t_i is the corresponding processing time. During scheduling, only the predicted class and its estimated demand e_c are available to the assignment policy; the actual processing time remains hidden.

3.3 Workload-Aware Server Assignment

Let

$$\mathcal{S} = \{S_1, S_2, \dots, S_m\} \quad (7)$$

denote the set of cloud servers. Each server has a finite active-task capacity $Q_{\max}$. For an arriving task at time t , server S_j is eligible if its current active-task count satisfies

$$q_j(t) < Q_{\max}. \quad (8)$$

The proposed scheduler maintains the predicted remaining workload $\hat{L}_j(t)$ of each server. Waiting tasks contribute their full predicted service demand, running tasks contribute only their estimated remaining demand, and completed tasks contribute zero load.

For a new task with predicted demand e_c , the scheduler selects

$$S^* = \arg \min_{S_j: q_j(t) < Q_{\max}} \left(\hat{L}_j(t) + e_c \right). \quad (9)$$

If multiple servers have the same projected workload, the scheduler first selects the server with the smallest active queue length. Remaining ties are resolved using a rotating fair-selection rule to avoid persistent preference for the same server. If all server queues are full, the incoming task is rejected.

Algorithm 3.3 summarizes the scheduling procedure.

[!t] ML-Based Workload-Aware Scheduling [1] Train Random Forest and compute class-load estimates e_0, e_1, e_2 each incoming query at time t Extract $X = (R, D, Q_r, S_c)$ and predict workload class c Set predicted demand $e_c \leftarrow \{S_j : q_j(t) < Q_{\max}\}$ $\mathcal{E} = \emptyset$ Reject task Compute $\hat{L}_j(t) + e_c$ for $S_j \in \mathcal{E}$ Select server with minimum projected workload Break ties using minimum $q_j(t)$ and rotating selection Assign task to the selected server

3.4 Simulation-Based Evaluation

The proposed scheduler is evaluated against Random and Round Robin under identical workloads across Low, Medium, High, and Overload scenarios, ensuring paired comparison. Performance is assessed using response delay, throughput, failed tasks, server utilization, and load-balance dispersion, with the experimental configuration and statistical procedure given in Section 4.1.

4 Performance Validation

4.1 Experimental Environment

The framework is implemented in Python using the configuration summarized in Table 2. The ML dataset contains 500 mixed-complexity analytical queries, split into 350 training and 150 testing queries. A separate pool of 4,000 newly generated queries is used for scheduling evaluation, ensuring that scheduling workloads are not drawn from the classifier training or test sets.

Low, Medium, and High classes are defined using training-only equal-frequency processing-time thresholds of 0.8013 s and 2.8888 s. The four scheduling scenarios use target offered-load ratios of $\rho = 0.40, 0.65, 0.85, 1.20$, yielding realized means of 0.4004, 0.6524, 0.8481, and 1.2001. For each paired run, all scheduling methods receive the same sampled tasks, arrival times, predicted workloads, and actual service requirements, ensuring a fair comparison.

4.2 Evaluation Metrics

Five metrics are used to evaluate scheduling performance. Let N_c denote the number of completed tasks, N_f the number of rejected tasks, m the number of servers, B_j the busy time of server j , and T_{makespan} the simulation makespan.

1. **Average Delay** (D_{avg}): Mean response time from task arrival to completion,

$$D_{\text{avg}} = \frac{1}{N_c} \sum_{i=1}^{N_c} \left(t_i^{\text{finish}} - t_i^{\text{arrival}} \right). \quad (10)$$

2. **Throughput** (T): Number of completed tasks per unit time,

$$T = \frac{N_c}{T_{\text{makespan}}}. \quad (11)$$

Table 2: Experimental Configuration

Parameter	Configuration
IoT dataset	5,000 records, 30 days, 10 locations, 5 sensors, 3 events
ML query set	500 queries (350 train, 150 test)
Scheduling query pool	4,000 independent queries
Workload features	R, D, Q_r, S_c
ML classifier	Random Forest, 100 trees
ML classes	Low, Medium, High
Servers (m)	10
Capacity ($Q_{\max}$)	15 active tasks/server
Target load (ρ)	0.40, 0.65, 0.85, 1.20
Inter-arrival gap (τ)	0.1360, 0.2579, 0.8184, 0.5797 s
Tasks per scenario	180, 180, 180, 1,036
Simulation runs	20 paired runs/scenario
Schedulers	Random, Round Robin, Proposed

3. **Failed Tasks** (N_f): Number of incoming tasks rejected because all server capacities are full.
4. **Average Utilization** (U_{avg}): Mean fraction of simulation time for which the servers are busy,

$$U_{\text{avg}} = \frac{1}{m} \sum_{j=1}^m \frac{B_j}{T_{\text{makespan}}} \times 100. \quad (12)$$

5. **Load-Balance Dispersion** (σ_U): Standard deviation of server utilization,

$$\sigma_U = \sqrt{\frac{1}{m} \sum_{j=1}^m (U_j - U_{\text{avg}})^2}. \quad (13)$$

Lower values of D_{avg} , N_f , and σ_U indicate better performance, whereas higher throughput and utilization are preferred. Results are reported as mean $\pm$ 95% confidence intervals over 20 paired runs. Proposed-versus-baseline comparisons use paired t -tests with Holm correction for multiple comparisons, with statistical significance assessed at an adjusted $p < 0.05$.

5 Results and Analysis

5.1 Classification Performance

The Random Forest classifier achieved an accuracy of 94.00% on the held-out test set, with a macro F1-score of 0.9361. Table 3 summarizes the class-wise performance. The High class achieved perfect recall, while the Medium class had the lowest recall at 0.8571. Most errors occurred between adjacent workload classes.

Table 3: Random Forest Classification Performance

Class	Precision	Recall	F1	Support
Low	0.8571	0.9730	0.9114	37
Medium	0.9796	0.8571	0.9143	56
High	0.9661	1.0000	0.9828	57

The confusion matrix in Fig. 2 contains 141 correct predictions out of 150 test samples. Six Medium queries were classified as Low and two as High, while only one Low query was classified as Medium. No High query was misclassified.

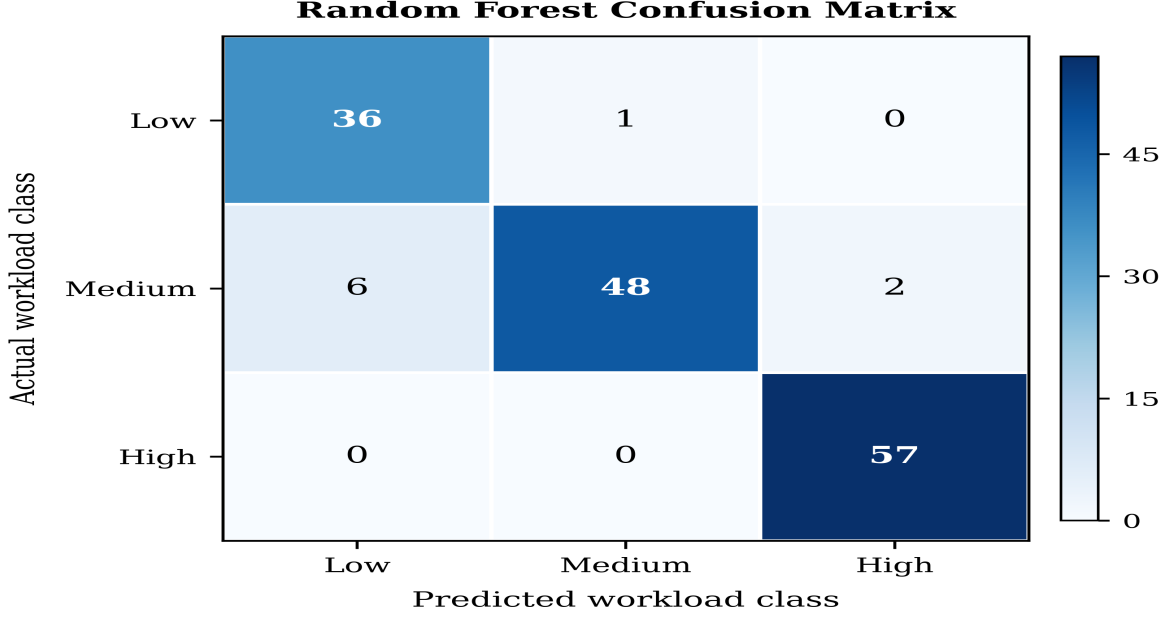


Figure 2: Confusion matrix of the Random Forest classifier on the held-out test set.

The feature-importance results in Fig. 3 identify estimated output cardinality R as the strongest predictor with an importance of 0.5415, followed by estimated scanned data volume D at 0.2735 and query range Q_r at 0.1814. Sensor coverage S_c contributes 0.0035. Since R and D are correlated workload descriptors, these impurity-based importance values are interpreted as descriptive rather than causal.

The classifier also achieved 93.53% accuracy on the independent 4,000-query scheduling pool, indicating consistent predictive performance beyond the original train-test query set.

5.2 Scheduling Performance

Tables 4 and 5 summarize the scheduling results over 20 paired runs. Random (R), Round Robin (RR), and Proposed (P) are evaluated using identical workload realizations. Results are reported as mean $\pm$ 95% CI.

Figs. 4 and 5 summarize the overall scheduling trends across Low, Medium, High, and Overload workloads. The detailed behavior of the schedulers is discussed in the following workload-specific subsections.

5.2.1 Low and Medium Workloads

Under Low load, Proposed and Round Robin achieve identical performance, indicating that workload-aware assignment offers little additional benefit when capacity is abundant; however,

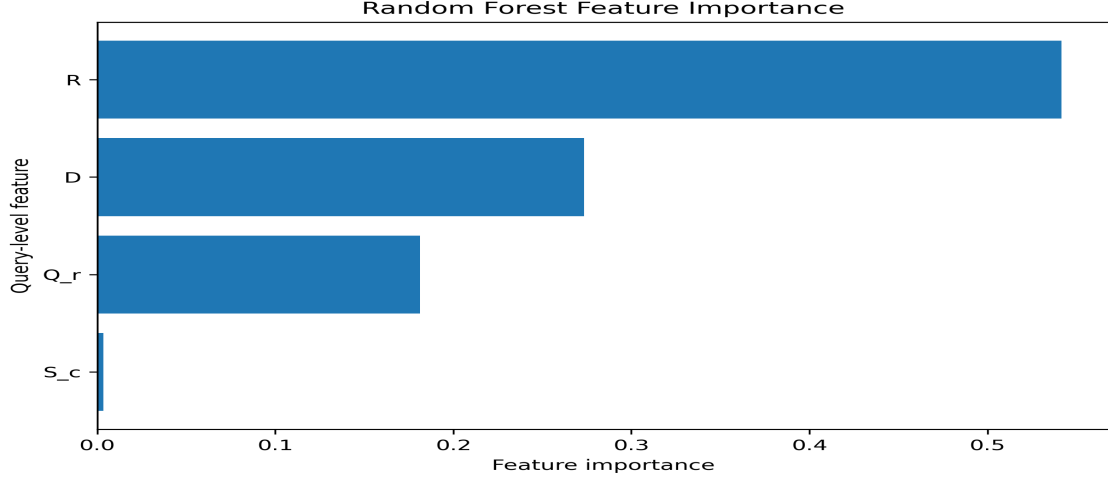


Figure 3: Random Forest feature importance for query workload classification.

Table 4: Scheduling Performance: Delay, Throughput, and Failures

Sc.	Alg.	Delay (s)	Thr. (t/s)	Failed
3*Low	R	0.685 ± 0.016	7.115 ± 0.052	0
	RR	0.545 ± 0.004	7.222 ± 0.012	0
	P	0.545 ± 0.004	7.222 ± 0.012	0
3*Med.	R	2.999 ± 0.130	3.451 ± 0.063	0
	RR	1.692 ± 0.020	3.734 ± 0.014	0
	P	1.683 ± 0.019	3.736 ± 0.013	0
3*High	R	17.117 ± 0.994	0.948 ± 0.029	0
	RR	8.389 ± 0.260	1.135 ± 0.010	0
	P	7.161 ± 0.118	1.163 ± 0.006	0
3*Over.	R	71.324 ± 0.992	1.376 ± 0.006	51.75 ± 2.69
	RR	63.919 ± 1.014	1.397 ± 0.008	35.35 ± 2.73
	P	63.192 ± 1.022	1.402 ± 0.008	33.45 ± 2.93

Proposed still outperforms Random by reducing delay by 20.48% and load-balance dispersion by 74.65%. Under Medium load, Proposed reduces delay from 1.6924 s to 1.6828 s relative to Round Robin, a 0.57% improvement that remains significant after Holm correction ($p < 0.001$), while its small gains in throughput, utilization, and load balance are not statistically significant after multiple-comparison adjustment.

5.2.2 High and Overload Workloads

Under High load, Proposed shows its strongest advantage, reducing delay by 14.64%, increasing throughput by 2.52%, increasing utilization by 2.53%, and reducing utilization dispersion by 47.59% relative to Round Robin, with all improvements remaining significant after Holm correction ($p < 0.001$). Compared with Random, it reduces delay by 58.17%, improves throughput by 22.70%, increases utilization by 22.78%, and lowers utilization dispersion by 75.58%, confirming that workload-aware assignment becomes more beneficial as server contention increases.

Under Overload ($\rho \approx 1.20$), Proposed rejects 33.45 ± 2.93 tasks per run, compared with 35.35 ± 2.73 for Round Robin and 51.75 ± 2.69 for Random. Relative to Round Robin, it reduces task rejection by 5.37% and delay by 1.14%, with both improvements remaining significant after Holm correction, while differences in throughput, utilization, and load balance are not significant.

Figure 6 highlights task rejection under the Overload scenario.

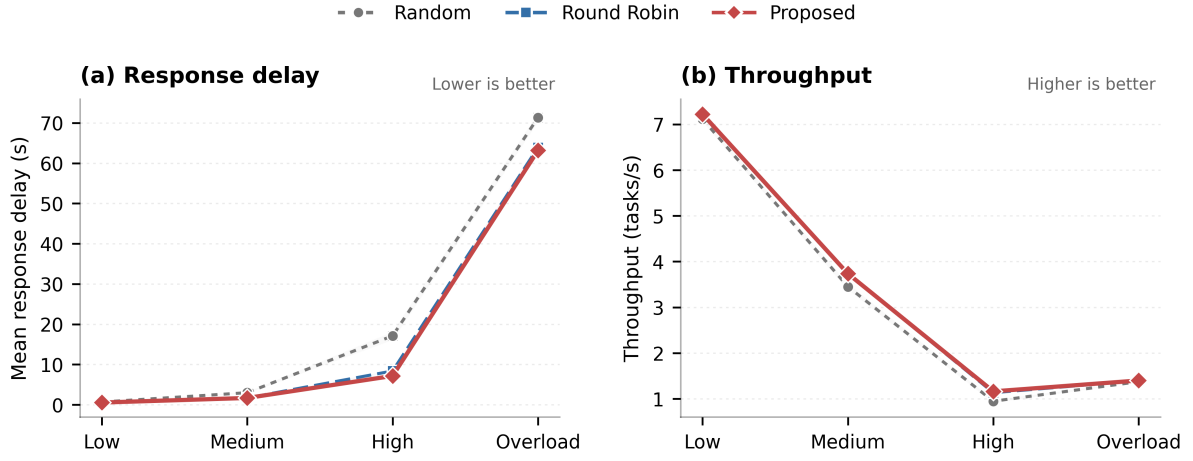


Figure 4: Scheduling performance across workload scenarios in terms of (a) response delay and (b) throughput. Error bars denote 95% confidence intervals over 20 paired runs.

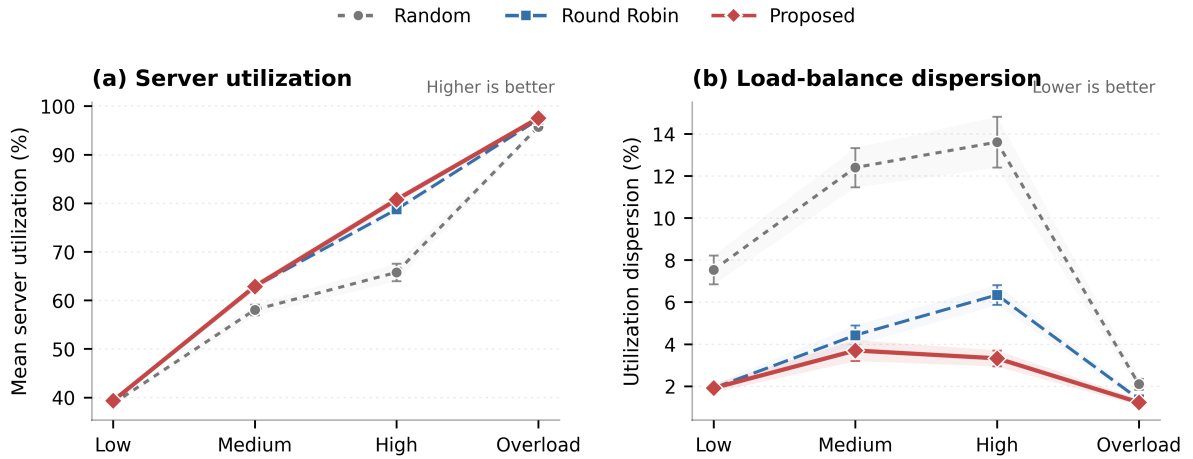


Figure 5: Resource-level behavior across workload scenarios in terms of (a) server utilization and (b) load-balance dispersion. Error bars denote 95% confidence intervals over 20 paired runs.

Table 5: Scheduling Performance: Utilization and Load Balance

Sc.	Alg.	Util. (%)	Balance Std. (%)
3*Low	R	38.74 ± 0.41	7.53 ± 0.69
	RR	39.32 ± 0.27	1.91 ± 0.20
	P	39.32 ± 0.27	1.91 ± 0.20
3*Med.	R	58.06 ± 1.10	12.40 ± 0.94
	RR	62.83 ± 0.74	4.43 ± 0.46
	P	62.87 ± 0.74	3.70 ± 0.50
3*High	R	65.75 ± 1.81	13.61 ± 1.21
	RR	78.74 ± 0.76	6.34 ± 0.47
	P	80.73 ± 0.76	3.32 ± 0.38
3*Over.	R	95.73 ± 0.35	2.10 ± 0.26
	RR	97.24 ± 0.37	1.38 ± 0.13
	P	97.51 ± 0.31	1.23 ± 0.15



Figure 6: Mean rejected tasks under the Overload scenario with 95% confidence intervals.

5.3 Discussion

The results show that workload-aware scheduling is more effective under higher levels of resource contention. Under Low load, the performance of the proposed and Round Robin is quite similar, and under Medium load, there is a slight improvement in delay, though not enough to be considered significant. The strongest gains are seen in the case of High load, where the remaining server workload to be predicted is more efficiently assigned across servers.

With Overload, capacity limits are the primary factor influencing performance; however, the proposed scheduler still reduces delay and task rejection compared to Round Robin. These results indicate that useful scheduling gains can be achieved even with imperfect workload classification, especially in the presence of high server contention.

6 Conclusion and Future Work

This study presents an ML-based simulation framework for IoT data-cube workload management in cloud data centers. The proposed scheduler reduces delay by up to 11.0%, improves throughput by up to 18.3%, and improves load balance by up to 74.2% over Random scheduling. Compared with Round Robin, load balancing improves by 23.0% under Medium workloads and by 13.7% under High workloads. Sensitivity and cross-seed analyses further demonstrate stable performance across varying configurations.

The study is limited by synthetic data, a simplified cloud environment, and evaluation with a single ML model. Future work will use real IoT datasets, evaluate additional classifiers, extend the framework to fog/edge environments, and investigate reinforcement learning and

multi-resource scheduling.

Conflict of Interest

The author has no conflict of interest in publishing this research.

Data Availability

No data is generated in this research.

References

- [1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
- [2] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
- [3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms,” *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.
- [4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- [5] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [6] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011, doi: 10.1002/spe.995.
- [7] T. D. Braun, H. J. Siegel, N. Beck, L. B. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, “A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems,” *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.
- [8] D. Saxena, J. Kumar, A. K. Singh, and S. Schmid, “Performance analysis of machine learning centered workload prediction models for cloud,” arXiv preprint arXiv:2302.02452, 2023.
- [9] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, “iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments,” *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.
- [10] D. N. Jha, K. Alwasel, A. Alshoshan, X. Huang, R. K. Naha, S. K. Battula, S. Garg, D. Puthal, P. James, A. Y. Zomaya, S. Dustdar, and R. Ranjan, “IoTSim-Edge: A simulation framework for modeling the behavior of Internet of Things and edge computing

- environments,” *Software: Practice and Experience*, vol. 50, no. 6, pp. 844–867, 2020, doi: 10.1002/spe.2787.
- [11] X. Zeng, S. K. Garg, P. Strazdins, P. P. Jayaraman, D. Georgakopoulos, and R. Ranjan, “IOTSim: A simulator for analysing IoT applications,” *Journal of Systems Architecture*, vol. 72, pp. 93–107, 2017.
 - [12] K. Alwasel, D. N. Jha, F. Habeeb, U. Demirbaga, O. Rana, T. Baker, S. Dustdar, M. Villari, P. James, E. Solaiman, and R. Ranjan, “IoTSim-Osmosis: A framework for modeling and simulating IoT applications over an edge-cloud continuum,” *Journal of Systems Architecture*, vol. 116, Art. no. 101956, 2021, doi: 10.1016/j.sysarc.2020.101956.
 - [13] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, “iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments,” *Journal of Systems and Software*, vol. 190, Art. no. 111351, 2022.
 - [14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
 - [15] J. R. Quinlan, “Induction of decision trees,” *Machine Learning*, vol. 1, pp. 81–106, 1986.
 - [16] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
 - [17] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, pp. 273–297, 1995.
 - [18] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, 2015.