# Data-priority Aware Fair Task Scheduling for Stream Processing at the Edge

Faiza Akram*, Peng Kang†, Palden Lama† and Samee U. Khan*
*Department of Electrical and Computer Engineering, Mississippi State University, Starkville, MS 39762, USA.
†Department of Computer Science, University of Texas at San Antonio, San Antonio, TX 78256, USA.

***Abstract*—Real-time data stream processing at the edge is crucial for time-sensitive tasks within large-scale IoT systems. Task scheduling plays a key role in managing the Quality of Service (QoS), necessitating a prioritization system to distinguish between high and low-priority tasks, thus ensuring efficient data processing on edge nodes. Existing scheduling algorithms rigidly prioritize tasks deemed as high-priority, often at the expense of fairness and overall system efficiency. In this paper, we propose a Priority-aware Fair Task Scheduling (FTS-Hybrid) algorithm that addresses these challenges by managing priority-based task execution in a controlled manner. Our task scheduling algorithm streamlines resource utilization and enhances system responsiveness, contributing to low latency and high throughput, outperforming competing techniques including First-Come-First-Serve (FCFS), Round Robin (RR), and Priority Scheduling (PS). We implemented FTS-Hybrid on Apache Storm and evaluated its performance using an open-source real-time IoT benchmark (RIoTBench). Experimental results show that the FTS-Hybrid algorithm reduces task execution latency by 24%, 31%, and 26% compared with FCFS, RR, and PS, respectively, by strategically mitigating queuing delays under dynamic workload conditions.**

***Index Terms*—Edge Stream Processing, Fair Scheduling, Steady State Queuing.**

## I. Introduction

The growth of data-intensive applications, including e-healthcare, home automation, smart cities, industrial operations, and intelligent driving, demands real-time data stream processing. This is crucial for prompt responses, particularly for urgent tasks within large-scale IoT systems. These time-sensitive applications require real-time response to events and cannot tolerate the delays incurred in sending data to the cloud for decision-making. The necessity for immediate decision-making on-site has prompted researchers to investigate edge computing's potential. By positioning services and functions closer to users, edge computing can enhance quality-of-service (QoS) through efficient data processing on edge servers [1].

QoS management in edge computing networks involves two critical variables: the execution order of tasks over time and the allocation of computing resources to prevent delays [2]. Given the resource scarcity of the edge environment, and the need to handle numerous tasks, it is desirable to serve tasks according to their priority [3]. For example in healthcare, tasks are classified as emergency or non-emergency, with emergency tasks taking precedence due to their urgency [4]. An efficient workload mix in edge computing is vital for dynamic applications like smart city monitoring and environmental sensing, ensuring prompt responses and optimized resource usage for effective city management amid evolving scenarios. Therefore, task scheduling has become a focal point of research, with various strategies available for both cloud and edge environments to prioritize tasks according to their importance [5]. However, existing scheduling algorithms often overlook fairness and overall system efficiency while rigidly prioritizing tasks that are of high importance [6].

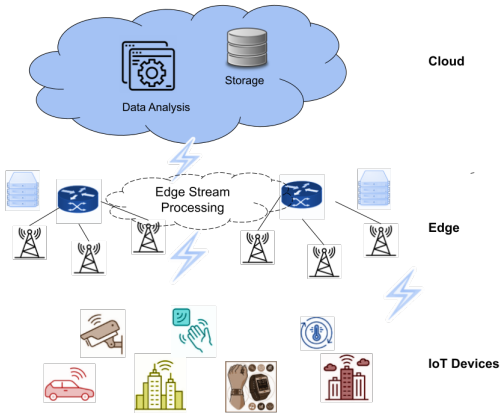



Fig. 1: Edge Stream Processing.

In this paper, we introduce a Priority-aware Fair Task Scheduling algorithm (FTS-Hybrid) in which tasks with high priority receive scheduling preference in a controlled manner to ensure fair scheduling across all priority levels. Our goal is to improve task execution latency across all priority levels by minimizing queuing delays and improving overall system efficiency. Experimental evaluation using an open-source real-time IoT benchmark, RIoTBench [7], and a popular stream processing engine, Apache Storm, demonstrate the effectiveness of FTS-Hybrid across high, medium, and low-priority tasks. Additionally, steady-state queuing analysis in Section III underscores its superiority over traditional methods.

Our key contributions are as follows:

- We introduce the Fair Task Scheduling algorithm (FTS-Hybrid), designed to prioritize high-priority tasks without neglecting fairness and overall system efficiency.
- We conduct a comparative analysis of our proposed scheduling algorithm against First-Come-First-Serve (FCFS), Round Robin (RR), and Priority Scheduling (PS). Results demonstrate improved latency and com-

parable throughput across all priorities.

- We perform steady-state queuing analysis across different algorithms to compare the average waiting time for tasks with different priorities.
- Our priority-aware FTS-Hybrid algorithm minimizes queuing delays and processing bottlenecks during latency spikes under dynamic workload conditions through intelligent task scheduling. Experiments results show latency reductions of 24%, 31%, and 26% compared to FCFS, RR, and PS, respectively.

The rest of the paper is structured as follows. Section II provides the background and motivation. Section III presents the design and steady-state queuing analysis. Section IV describes the testbed setup and experimental evaluation. Related work is discussed in Section V. Section VI summarizes the findings, conclusions, and future work.

## II. Background & Motivation

### A. Stream Processing System at the Edge

As shown in Figure 1, Stream Processing Engines (SPEs) are increasingly employed on IoT Gateways and edge routers for data processing, capitalizing on the low latency afforded by their closeness to users and IoT devices. These edge nodes offer improved computational capabilities over wireless sensor networks, yet they are limited compared to the resources available in the Cloud [8]. The SPEs receive data streams from IoT devices and run domain-specific applications for data processing. Each application consists of a Directed Acyclic Graph (DAG), or topology, comprising spouts and bolts. Spouts read data from external sources and emit tuples (basic data units) into the topology for processing. Bolts serve as data processing units/operators, enabling a range of tasks from basic filtering to advanced operations like Machine Learning (ML)-based classification algorithms. Notable streaming processing frameworks include Apache Storm, Apache Spark, Flink, etc. In this paper, we use Apache Storm as it supports real-time data processing of individual data tuples with minimal delay.

### B. Task Priority in Stream Processing System

In the context of edge stream processing, we define a task as an operation performed on a single data tuple. By default, all SPEs including Apache Storm process data tuples in the order of their arrival. However, this can lead to unacceptable queuing delays for high-priority tasks as shown in Figure 2. Scheduling tasks based on their priority is crucial for QoS management in a resource-constrained edge environment. Task priorities can be defined by users based on time-sensitivity of the data tuples that need to be processed. However, strict priority based scheduling can lead to starvation of lower priority tasks and inefficient resource utilization, specially under dynamic workload conditions. Therefore, there is a need to devise a fair scheduling strategy that strikes a balance between task prioritization and fairness to improve overall system efficiency.

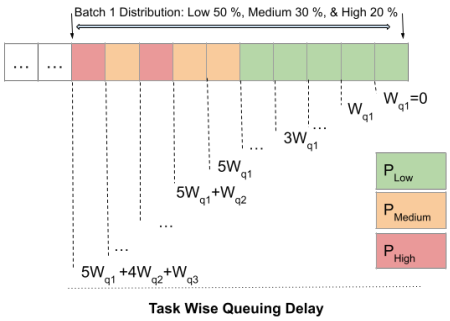



Fig. 2: Input Data Stream with different priorities. $W_{q_1}$ ,$W_{q_2}$, $W_{q_3}$ are expected waiting times in the queue for low, medium and high priority tasks if they are processed in the order of arrival.

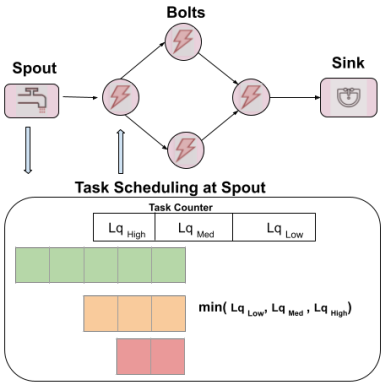



Fig. 3: FTS-Hybrid task scheduling for edge stream processing. A batch of tasks are scheduled based on their priority and queue length. $Lq_1$, $Lq_2$, $Lq_3$ are the queue lengths for low, medium and high priority tasks

## III. System Design

In this section, we describe our Priority-Aware Fair Task Scheduling algorithm and present a steady-state queuing analysis of our edge stream processing system.

### A. Priority-Aware Fair Task Scheduling

As shown in Figure 3, our FTS-Hybrid scheduling technique places incoming data tuples into separate queues based on their priority. We categorize tasks into low, medium, and high. The scheduling algorithm is shown in Algorithm 1. For every batch of input tuples (with fixed batch size), the spout is customized to emit a minimum number of tuples from each queue starting from the high-priority queue and moving on to medium and low-priority queues. The number of tuples emitted is dictated by the shortest queue length measured among the non-empty priority queues for a given batch. This process is repeated until all tuples in the batch are emitted to downstream operators (bolts) in the application topology. For example, given a batch size of 10 with a task distribution of 5, 3, and 2 tuples for low, medium, and high-priority tasks respectively, two tuples are emitted from each queue since the shortest queue length is 2.

**Algorithm 1:** FTS-Hybrid task scheduling.

Let $T'$ denote a stream of tuples with different priorities coming to an application
$B$ is a batch of tuples to be scheduled
$Q_{\text{low}}$ is an empty queue for low-priority data
$Q_{\text{med}}$ is an empty queue for medium-priority data
$Q_{\text{high}}$ is an empty queue for high-priority data
**for** $i = 1$ to $B$ **do**
    Inspect priority $p_{t_i}$ of incoming data tuple $t_i$
    **if** $p_{t_i} = 1$ **then**
        Enqueue $t_i$ into $Q_{\text{low}}$
    **else if** $p_{t_i} = 2$ **then**
        Enqueue $t_i$ into $Q_{\text{med}}$
    **else if** $p_{t_i} = 3$ **then**
        Enqueue $t_i$ into $Q_{\text{high}}$
    **end if**
**end for**
$L_{q_{\text{low}}} \leftarrow \begin{cases} \infty & \text{if } Q_{\text{low}} = \varnothing \\ \text{Length of } Q_{\text{low}} & \text{otherwise} \end{cases}$
$L_{q_{\text{med}}} \leftarrow \begin{cases} \infty & \text{if } Q_{\text{med}} = \varnothing \\ \text{Length of } Q_{\text{med}} & \text{otherwise} \end{cases}$
$L_{q_{\text{high}}} \leftarrow \begin{cases} \infty & \text{if } Q_{\text{high}} = \varnothing \\ \text{Length of } Q_{\text{high}} & \text{otherwise} \end{cases}$
**while** $Q_{\text{low}} \neq \varnothing$ or $Q_{\text{medium}} \neq \varnothing$ or $Q_{\text{high}} \neq \varnothing$ **do**
    MinSize $= \min\{L_{q_{\text{low}}}, L_{q_{\text{med}}}, L_{q_{\text{high}}}\}$
    **for** $i = 1$ to MinSize **do**
        Dequeue a tuple from $Q_{\text{high}}$ and emit
    **end for**
    **for** $i = 1$ to MinSize **do**
        Dequeue a tuple from $Q_{\text{med}}$ and emit
    **end for**
    **for** $i = 1$ to MinSize **do**
        Dequeue a tuple from $Q_{\text{low}}$ and emit
    **end for**
**end while**

Updating the queues, the highest priority queue is empty now, leaving 3 tuples in the low and 1 tuple in the medium priority queue. Since now the shortest queue length will be 1 hence, one tuple is emitted from the medium and low-priority queues. Finally, the remaining tuples in the low-priority queue are emitted. By processing groups of tuples within a batch in this manner, we aim to strike a balance between task prioritization and fairness across all priority levels.

### *B. Steady State Queuing Analysis*

We have implemented FTS-Hybrid on Apache Storm, configured on a single machine to emulate an IoT Gateway, running the RIoTBench [7] application. Further implementation details can be found in Section IV-A. In this section, we perform a steady-state queuing analysis to understand the long-term behavior of our edge stream processing system. We employ the $M/M/1$ queuing model where both interarrival times (the time between successive task arrivals) and service times (task execution time) follow exponential distributions.

TABLE I: Mathematical Notations used in Queuing Analysis

| Notation | Description |
|---|---|
| $Q_H$ | average number of tuples in the queue for high-priority |
| $s_H$ | service time for high-priority |
| $Q_M$ | average number of tuples in the queue for medium-priority |
| $s_M$ | service time for medium-priority |
| $Q_L$ | average number of tuples in the queue for low-priority |
| $s_L$ | service time for low-priority |
| $\lambda_i$ | arrival rate for each-priority |
| $R$ | Total residence time of a task in the queue |

TABLE II: Analysis of Average residence time in queue

| | Low | Medium | High |
|---|---|---|---|
| FCFS | 5.7 ms | 90 ms | 10.5 ms |
| RR | 13.67 ms | 13.67 ms | 13.67 ms |
| PS | 35.9 ms | 35.9 ms | 3 ms |

The total residence time $R$ for a task of specific non-preemptive priority(high, medium, low)in the system following **PS** is calculated as:

$$\begin{aligned} R_H &= Q_H s_H + s_H \\ R_M &= Q_H s_H + Q_L s_L + (R_H - s_H)U_H + s_M \\ R_L &= Q_H s_H + Q_M s_M + (R_H - s_H)(1 - U_H) \\ &\quad + (R_M - s_M)(1 - U_M) + s_L \end{aligned} \tag{1}$$

Here, $Q_H s_H$, $Q_M s_M$, $Q_L s_L$ are the additional delays due to the processing of high, medium, and low-priority tasks respectively. The fraction of time the server is busy with a particular priority is

$$U_i = \lambda_i \times s_i, \quad \forall i \in \{H, M, L\} \tag{2}$$

The total residence time $R$ for a task of specific priority in the system following **RR** is calculated as:

$$\begin{aligned} R_H &= \frac{Q_H s_H + Q_M s_M + Q_L s_L}{3} \\ R_M &= \frac{Q_M s_M + Q_L s_L + Q_H s_H}{3} \\ R_L &= \frac{Q_L s_L + Q_H s_H + Q_M s_M}{3} \end{aligned} \tag{3}$$

The total residence time $R$ for a task of specific priority in the system following **FCFS** is calculated as::

$$\begin{aligned} R_H &= \frac{U_M U_H s_1 + U_M s_2 + (1 - U_H)s_1}{1 - U_H - U_M U_H^2} \\ R_M &= \frac{U_L U_H s_1 + U_L s_3 + (1 - U_H)s_1}{1 - U_H - U_L U_H^2} \\ R_L &= \frac{U_H U_M s_2 + U_H s_1 + (1 - U_M)s_2}{1 - U_M - U_H U_M^2} \end{aligned} \tag{4}$$

First, We analyze the effectiveness of three alternative methods: i) **FCFS**, as the default in Apache Storm, causes delays for high-priority tasks due to its disregard for priorities; (ii) **RR**, aims for fairness RR treats all tasks equally regardless of

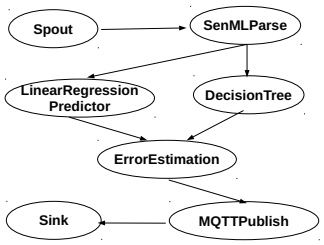



Fig. 4: Layout of PRED topology.

priority (iii) **PS**, implements strict priorities, processing higher-priority tasks before lower-priority ones within data batches. However, both PS and RR have drawbacks in prioritization. To address this, the FTS-hybrid algorithm blends RR's fairness with PS's priority focus, minimizing wait times for low and medium priorities while maintaining timely high-priority task processing. Experimental results support FTS-hybrid's effectiveness in achieving this balance (Section IV). For theoretical analysis in batch of size 10 we consider $Q_H$ = 2 $Q_M$= 3, and $Q_L$ = 5, $s_H$ = 1ms, $s_M$ = 3ms, and $s_L$= 6ms, $U_H$= 0.2, $U_M$= 0.3, $U_L$= 0.5 .Table II compares the average residence times estimated for tasks with different priorities.

## IV. Evaluation

### A. Experimental TestBed

*1)* ***Edge node configuration****:* We set up a prototype testbed using a single VM equipped with 4 CPU cores and 8 GB RAM to replicate an IoT Gateway running Ubuntu 20.04.6 LTS. The VM was hosted in NSF-supported Chameleon Cloud [9]. We used Apache Storm (Version 2.1.0) to serve as the edge stream processing engine.

*2)* ***Benchmark****:* We evaluate our proposed method using the RIoTBench benchmark [7] suite, comprising four applications: Extract-Transform-Load (ETL), Model Training (TRAIN), Statistical Summarization (STATS), and Predictive Analytics (PRED). Due to the space limitation, we only present our results obtained for the PRED application. Figure 4 shows the application topology.

*3)* ***Dynamic Workload Generation****:* We employed the New York City Taxi Trips (TAXI) dataset to create a workload that accurately reflects real-world situations. The dataset consists of smart transportation data derived from 2 million taxi trips taken in New York City in 2013. Each trip record encompasses details like pickup and drop-off dates, taxi and license information, start and end coordinates with timestamps, metered distance, and taxes and tolls paid. To generate a dynamic workload, we adapted RIoTBench's input generator function to produce varying workload intensities.

### B. Performance Analysis under Dynamic Workload

For performance evaluation, we measured the 95th-percentile end-to-end latency and throughput of data tuples flowing through the stream processing topology under a dynamic workload. We measured throughput by counting the number of tuples that reach the sink of the application topology, shown in Figure 4. We measured latency by assigning each tuple a unique ID and comparing timestamps at the source and the same sink used for the throughput measurement. The performance was periodically measured at one-minute sampling intervals. Furthermore, we analyzed the impact of varying the proportion of low, medium, and high-priority tasks on latency and throughput. Three scenarios were explored: 70% low, 20% medium, and 10% high-priority tasks; 50% low, 30% medium, and 20% high-priority tasks; and equal distribution of 33% for each priority level. The first two scenarios mirror real-world situations where urgent tasks are relatively scarce compared to non-urgent tasks. The final scenario was selected as an extreme and improbable situation to test the limitations of the system.

TABLE III: Percentage improvement in latency due to FTS-Hybrid for different workload distributions.

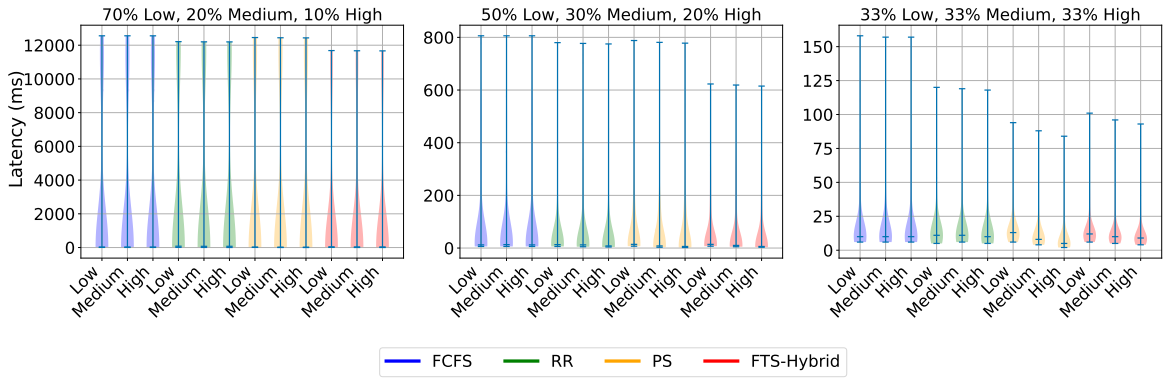
| FTS-Hybrid vs. | FCFS | RR | PS |
|---|---|---|---|
| 70% Low, 20% Medium, 10% High | | | |
| High-priority | 24.2% | 31.05% | 25.8% |
| Medium-priority | 24.1% | 31.03% | 25.7% |
| Low-priority | 23.8% | 30% | 25.7% |
| 50% Low, 30% Medium, 20% High | | | |
| High-priority | 33.34% | 4.7% | 3.4% |
| Medium-priority | 37.1% | 4% | 2.4% |
| Low-priority | 40.74% | 4% | 1.9% |
| 33% Low, 33% Medium, 33% High | | | |
| High-priority | 12.39% | 14.12% | 10.1% |
| Medium-priority | 22.8% | 22.6% | -5.09% |
| Low-priority | 30.4% | 27.13% | -15.58% |

Our findings, depicted in Fig.5a and Fig.5b, reveal the influence of workload distribution on 95th percentile latency and throughput, respectively, under conditions of a batch size of 100. Notably, our algorithm effectively addresses bottlenecks during latency spikes by optimizing task execution by using minimum batch from low-priority tasks as well resulting in efficient processing and reduced wait times across all task priorities. This improvement is attributed to our algorithm's adept management of queue length while scheduling tasks, crucial for mitigating latency spikes during peak loads. However, the proposed approach struggles with an equal workload mix since all queues increase at the same pace, rendering the shortest queue strategy ineffective. PS works better because it constantly prioritizes high-priority activities regardless of queue length.

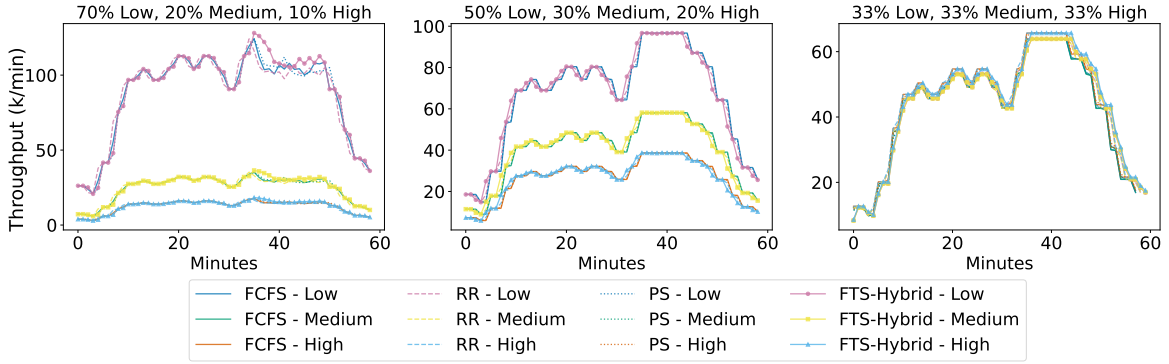
Table III depicts the percentage reduction in latency due to FTS-Hybrid over competing scheduling algorithms. In the majority of applications, high-priority tasks are typically sparse. Workloads in many applications are dynamic and are subject to variation over time. Our algorithms also achieve improvement of approx. 2% in throughout.

### C. Batch Sensitivity Analysis

Batching data into larger transactions can notably boost performance, particularly in high-latency scenarios, depending on available network resources. While smaller batch sizes en-




(a) Violin plots illustrate the distribution of 95th percentile latency, captured at one-minute intervals across a span of one hour.




(b) Throughput.

Fig. 5: The effect of varying the proportion of low, medium, and high-priority tasks on task execution latency and throughput. A batch size of 100 was used.

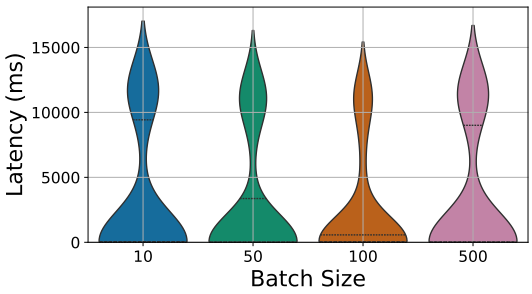



Fig. 6: Impact of batch size on the 95th percentile latency distribution for high-priority tasks utilizing the FTS-Hybrid algorithm. The workload mix comprises 70% low-priority, 20% medium-priority, and 10% high-priority tasks.

able faster processing and frequent system updates, excessively small batches may render priority scheduling ineffective. Conversely, larger batches may introduce delays due to increased processing requirements. Selecting the appropriate batch size is crucial. Our experimental results in Fig. 6, demonstrate that increasing the batch size from 10 to 100 can improve high-priority latency by up to 30% moving beyond this batch size increases the latency. While our focus is on high-priority data, similar trends apply to low and medium-priority data. We opted for a batch size of 100 as it strikes a balance between prioritization and responsiveness. This can be observed in Fig.6 by looking at the horizontal median latency line in violin graphs. The median latency range for batch size 100 lies at $\approx$ 2000 ms whereas for batch size 500 the range goes to $\approx$ 8000 ms. Nonetheless, in stream processing, batches may not always reach maximum size before processing begins.

## V. Related Work

Edge computing uses priority-based fair scheduling to efficiently process tasks in hardware-constrained systems. Rodrigo et al. [10] proposed a decentralized fair share prioritization method, adjusting job scheduling based on user quotas and past resource usage. Chai et al. [11] crafted a dynamic priority-based computation scheduling algorithm, prioritizing task processing performance and fairness. Similarly, Paymard et al. [12] proposed a scheme to jointly optimize computation and communication resource allocation to maximize profit and satisfy QoS requirements. Gao and Moh [13] proposed a joint computation offloading and prioritized task scheduling scheme, considering energy minimization and dynamic threshold setting. Madej et al. [6] presented different scheduling techniques, including a hybrid strategy that accounts for fairness among clients and job priorities in edge computing. Sev-

eral studies, such as those by [2], and [14], delve into the development of priority-aware task scheduling algorithms. These algorithms prioritize tasks based on urgency and appropriately offload data. However, it's worth noting that these scheduling techniques do not take into account batch processing. Also, they do not discuss the impact of varying data ratios for different priorities. One of the most relevant works we found in [15], EdgeWise enhances Stream Processing Engine (SPE) performance by integrating a congestion-aware scheduler with a fixed-size worker pool, boosting throughput and reducing delay. However, data priorities and the impact of different data ratios are not addressed.

Furthermore, innovative task scheduling schemes based on Bayesian classifiers have been devised to classify tasks into priority categories and schedule them accordingly, leading to enhanced completion time and throughput rates [16]. Shahid et al. [17] proposed a scheduler that organizes data based on the priority level and applies Bayesian optimization to achieve an SLO violation of 0% for high-priority data. In [18], authors propose a task-scheduling and resource allocation method that prioritizes work based on emergency levels derived from data collected from patient wearables.

Our approach effectively manages priority-based systems, preventing extended wait times for low-priority tasks. Dynamic emission order adjustment and tuple batching ensure fair task scheduling and mitigate switching costs, crucial for scenarios with medium to low-priority users.

## VI. Conclusion & Future Work

In this paper, we designed the FTS-Hybrid scheduling algorithm, focusing on enhancing Quality of Service (QoS) by improving throughput and latency for tasks of varying priorities. Our prototype was implemented using Apache Storm and Chameleon Cloud Server. We assessed its performance using RIoTBench, an open-source real-time IoT benchmark. In comparison to state-of-the-art techniques, which often overlook low and medium-priority tasks, our algorithm targets comprehensive improvement across all priority levels. Experimental results confirmed significant enhancements in both latency and throughput, marking a notable advancement of QoS in Edge computing. In the future, we aim to enhance FTS-Hybrid by monitoring the ratio of high-priority tasks in the workload and dynamically switching to PS scheduling when necessary. This adaptive approach will allow the system to respond effectively to fluctuations in task priorities, ensuring that critical tasks are prioritized during periods of increased demand. This adaptability ensures that system performance remains optimal under varying workload conditions. Furthermore, we also plan to predict the incoming workload intensity and use machine learning models to dynamically optimize the suitable batch size based on a feedback mechanism to further improve our results.

## Acknowledgment

This research is supported by National Science Foundation grants CNS 1911012, CNS 2124908, and CCF 2135439. Opinions expressed are those of the author(s) and do not necessarily reflect NSF's views.

## References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, 2016.

[2] M. S. Bali, K. Gupta, D. Gupta, G. Srivastava, S. Juneja, and A. Nauman, "An effective technique to schedule priority aware tasks to offload data on edge and cloud servers," *Measurement: Sensors*, vol. 26, 2023.

[3] D. M. Dakshayini and D. H. Guruprasad, "An optimal model for priority based service scheduling policy for cloud computing environment," *International Journal of Computer Applications*, 2011.

[4] H. Wang, J. Gong, Y. Zhuang, H. Shen, and J. Lach, "Healthedge: Task scheduling for edge computing with health emergency and human behavior consideration in smart homes," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 1213–1222, IEEE, 2017.

[5] T. Choudhari, M. Moh, and T.-S. Moh, "Prioritized task scheduling in fog computing," ACMSE '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[6] A. Madej, N. Wang, N. Athanasopoulos, R. Ranjan, and B. Varghese, "Priority-based fair scheduling in edge computing," in *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, 2020.

[7] A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: A real-time iot benchmark for distributed stream processing platforms," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, 2017.

[8] J. Wu, J. Hu, W. Shao, H. Chen, M. Gao, B. Chen, and J. Wu, "Maximum tolerable-delay redistribution approaches with network resource scheduling in edge-cloud elastic optical networks," in *2022 Asia Communications and Photonics Conference (ACP)*, IEEE, 2022.

[9] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, "Lessons learned from the chameleon testbed," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*, USENIX Association, July 2020.

[10] G. P. Rodrigo, P.-O. Östberg, and E. Elmroth, "Priority operators for fairshare scheduling," in *Job Scheduling Strategies for Parallel Processing* (W. Cirne and N. Desai, eds.), (Cham), pp. 70–89, Springer International Publishing, 2015.

[11] R. Chai, M. Li, T. Yang, and Q. Chen, "Dynamic priority-based computation scheduling and offloading for interdependent tasks: Leveraging parallel transmission and execution," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 10, pp. 10970–10985, 2021.

[12] P. Paymard, N. Mokari, and M. Orooji, "Task scheduling based on priority and resource allocation in multi-user multi-task mobile edge computing system," in *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pp. 1–7, 2019.

[13] L. Gao and M. Moh, "Joint computation offloading and prioritized scheduling in mobile edge computing," in *2018 International Conference on High Performance Computing and Simulation (HPCS)*, 2018.

[14] S. M. Salman, A. V. Papadopoulos, S. Mubeen, and T. Nolte, "Evaluating dispatching and scheduling strategies for firm real-time jobs in edge computing," in *IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1–6, 2023.

[15] X. Fu, T. Ghaffar, J. C. Davis, and D. Lee, "EdgeWise: A better stream processing engine for the edge," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, (Renton, WA), pp. 929–946, USENIX Association, July 2019.

[16] Y. Wang and S. Liu, "Task scheduling scheme for mobile edge computing via bayesian classifier," *Journal of Physics: Conference Series*, vol. 2504, p. 012041, may 2023.

[17] A. Shahid, P. Kang, P. Lama, and S. U. Khan, "Some new observations on slo-aware edge stream processing," in *2023 IEEE Cloud Summit*, pp. 27–32, 2023.

[18] Z. Sharif, L. Tang Jung, M. Ayaz, M. Yahya, and S. Pitafi, "Priority-based task scheduling and resource allocation in edge computing for health monitoring system," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 2, pp. 544–559, 2023.