Server Time Reservation for Periodic Real-Time Applications

Ali Balador
Ericsson Research
Kista, Sweden

Lizzy Tengana
Ericsson Research
Kista, Sweden

Mohammad Ashjaei
Mälardalens University
Västerås, Sweden

Raihan Ul Islam
Ericsson Research
Kista, Sweden

Saad Mubeen
Mälardalens University
Västerås, Sweden

ABSTRACT

To utilize edge and cloud in real-time industrial applications, communication with the edge and cloud servers should be predictable in timing. However, the predictability of offloading from device to servers cannot be guaranteed in an environment where multiple devices compete for the same edge and cloud resources due to potential server-side scheduling conflicts. To the best of our knowledge, the state-of-the-art lacks a technique for offloading real-time applications from multiple devices to a set of heterogeneous edge/cloud servers. To this end, this paper proposes a centralized resource reservation technique that enables the offloading of real-time applications to the edge and cloud in a predictable time-schedule. The proposed technique enables end-devices to request the server's time for offloadable real-time applications in advance, allowing a designated offloading server that guarantees the tasks' timely execution. Furthermore, the proposed technique is capable of optimizing the reservation scheduling strategy with the goal of minimizing the energy consumption of edge servers while meeting the stringent timing requirements of real-time applications. The results showed that the number of deadline satisfied jobs improved by 65%, and total energy consumption by 3%, compared to the second best algorithm among the ones that have been compared with the proposed algorithm when the number of jobs is changed.

CCS CONCEPTS

• **Computing methodologies** → **Self-organization**; • **Computer systems organization** → **Real-time system architecture**.

KEYWORDS

Computational Offloading, Real-Time Cloud, Resource Reservation

1 INTRODUCTION

As the world moves towards the fifth generation of network connectivity (5G) and computing services become closer to users, the digitalization opportunities promise to revolutionize even the most traditional industries. This increased network capacity together with diverse on-demand computing platforms (e.g., Cloud and Edge Computing) creates opportunities to enable resource-intensive industrial applications that were not feasible before, including infrastructure monitoring, smart manufacturing, and collaborative robots [7].

The cloud computing paradigm offers support for a wide range of computation-intensive applications. Although cloud computing provides enhanced storage and computing capacity, it can cause high and unpredictable latencies [9, 11]. Edge computing is a distributed computing paradigm and ecosystem, where the execution environment (e.g., computing and storage resources) is located closer to the data source compared to the traditional cloud computing paradigm. Edge computing also enables offloading of certain functionality from resource-constrained devices to edge and cloud servers. By offloading, it is possible to improve the performance of the local device, e.g., reduce energy consumption and extend its capabilities by accessing advanced services only possible to run in the edge-cloud [3]. Certainly, edge computing has lower latencies compared to the cloud; however, these latencies are still not consistent enough to meet the timing requirements of time-critical applications.

Supporting timing predictability of offloading in an environment where multiple devices compete for the same edge and cloud resources due to potential server-side scheduling conflicts is still an open challenge. To provide real-time computing as a service at the edge/cloud would require the optimization of the decision of where to offload, along with the determination of the best allocation of communication and computation resources as in general-purpose offloading schemes [5]. This paper focuses on the offloading decision problem of *"where to offload a time-critical application?"*. Real-time tasks and jobs can be classified as periodic or aperiodic. A periodic task execution repeatedly at regular intervals. However, executions of aperiodic tasks are requested unpredictably. The time interval between the arrivals of two consecutive requests in a periodic task is called its period. In this paper, we focus only on periodic tasks and propose a solution considering specific charactristic of periodic tasks. Apart from that, the system model in this paper does not consider communication between the device and the edge layers, and also at the edge.

In this paper, we propose a server time reservation technique for periodic real-time tasks to be offloaded to the edge. The proposed technique allows end devices to request external resources in advance for tasks that will be generated in the future; therefore, when such a task is created, it already has a designated offloading server that guarantees its timely execution. The main advantage of the proposed technique is that when no server is able to accept a reservation, the reservation is rejected, and since the actual job is an event in the future, the device still has time to allocate resources to execute the task locally. This reservation system is capable of optimizing the reservation scheduling strategy with the goal of minimizing energy consumption on the server side while meeting the stringent timing requirements of the applications.

The remainder of this paper is organized as follows. Section II discusses related work; and Section III presents the system model.

Section IV shows the offloading orchestrator, and the algorithms proposed in this paper were explained in Section V. Section VI shows the simulation configuration and results. Section VII concludes the paper and discusses future work.

2 RELATED WORK

There are several existing works that provide various offloading schemes to optimize the migration and scheduling of tasks from end-devices to general-purpose edge/cloud servers [3, 6]. These works mainly focus on minimizing energy consumption and delays. Resource reservation for computation offloading has also received attention in the state of the art. For instance, Kim et al. [4] study the optimization of communication resources, particularly the network access reservation when offloading data with high volume. The authors propose a protocol to manage network access reservation and encapsulate partitioned data offloading.

Toma et al. [10] use the total bandwidth server which claims the free background time of periodic tasks to schedule aperiodic tasks that are offloaded to servers. The offloading decisions of what and where to offload are made on the client-side employing greedy and dynamic programming algorithms.

Zhang et al. [12] predict the computation demand for offloading in mobile environments using Deep Learning with historical data. They present a reservation strategy for edge servers' resources based on the initial prediction. The resource scheduling and coordination is done by a server in the cloud layer, which sends reservation requests to bind geographical regions to nearby edge servers based on a resource demand prediction. The goals of this reservation scheduling is to achieve an efficient utilization of edge resources.

Regarding the problem of offloaded task scheduling on the server-side, it involves the allocation of resources following given scheduling goals [2]. Misra et al. [8] propose *Detour*, an offloading scheme for software-defined fog networks where devices locally decide which server to use for offloading by choosing the one with the shortest waiting and execution time. Then, the optimal network path to offload is selected. Azizi et al. [1] study the scheduling of offloaded tasks from multiple internet of things (IoT) devices to heterogeneous fog networks. The proposed algorithm is based on a randomized greedy strategy that can balance the energy consumption of the system and tasks' deadline satisfaction.

Our proposed algorithm transforms the server resource reservation problem into a scheduling problem where devices submit reservation requests ahead of time to ensure that the execution of real-time tasks can meet the timing requirements.

3 SYSTEM MODEL

In an industrial environment consisting of multiple real-time devices and edge/cloud servers, the possibility of offloading real-time tasks raises the question of what is the best way of distributing these tasks among the available servers without having scheduling conflicts. In this section, we describe our centralized server time reservation system aimed at optimizing the distribution of tasks among servers with the goal of minimizing energy consumption while meeting task deadlines.

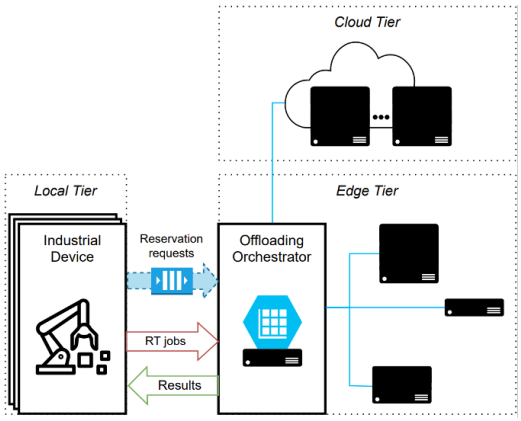


Figure 1: Device-edge-cloud architecture.

For clarity, each instance of a periodic task created by a device will be called a job, and it is assumed that each job can be offloaded independently.

The benefit of working with periodic tasks is that devices can anticipate the amount of resources that they will need to execute task instances (jobs) periodically in the future.

3.1 Job Model

Let a periodic task $RT_x = \{x^1, ..., x^m\}$ be a collection of jobs where a single job is defined as follows:

$$x^i = \{ps_{x^i}, w_{x^i}, d_{x^i}, dt_{x^i}, tout_{x^i}, in_{x^i}, out_{x^i}\}, \quad (1)$$

where ps_{x^i} is the starting time of a job at the Offloading Orchestrator (OO) server, w_{x^i} is the worst-case execution time in the local device, d_{x^i} is the deadline for job completion such that $d_{x^i} \leq ps_{x^{i+1}}$, dt_{x^i} is the deadline type of the job that is either hard or soft, $tout_{x^i}$ is the timeout of the job, in_{x^i} is the size of the input data in bytes and out_{x^i} is the size of the output data in bytes.

3.2 Device-edge-cloud Architecture

We consider a multi-tier architecture that contains local, edge, and cloud tiers. Each tier contains heterogeneous processing platforms that can be utilized to perform various types of tasks on different processors. This multi-tier architecture is suitable for applications that contain time-critical as well as non-time-critical but resource-intensive tasks. Time-critical tasks can be executed on the edge, while non-time-critical tasks can be executed on the cloud [5]. Fig. 1 shows a high-level overview of the interactions of our reservation technique in the multi-tier architecture. Devices in the local tier can send reservation requests to schedule the offloading of jobs in the edge and cloud tiers.

The details of our reservation technique within this multi-tier architecture are presented below.

- *Local device tier:* Industrial devices in the local tier generate periodic tasks with hard and soft deadlines.
  A device autonomously decides whether to execute a job locally or offload it to an external processor based on a utility function [8]. A utility function could consider different

criteria, such as job characteristics, device internal status, and edge/cloud status. If a device identifies that it may benefit from offloading a job, it can reserve processing capacity in an external server by sending a reservation request to the OO, a centralized entity at the edge tier that acts as a gateway to all the available edge and cloud servers. If a job could not be scheduled by OO, the device is quickly notified and possibly still has time to allocate local resources to execute the job without missing its deadline.

- *Edge tier:* The edge level consists of a distributed network of edge servers with heterogeneous computing capabilities, which are all connected to the OO server. The OO server receives reservation requests from devices in its reservation queue, then, the scheduling of jobs to servers at specific time slots becomes an optimization problem. The OO server computes a solution to the optimization problem that best meets the objectives of minimizing the energy consumption of the servers and the number of missed deadlines for the job. The OO server can accept or reject a reservation request accordingly and notify the device of the decision.
- *Cloud tier:* The cloud tier consists of a set of resource-rich servers with high computing power and storage capacity. However, due to the considerable physical distance between the servers and devices, processing of offloaded jobs in this environment may suffer from high and unpredictable latencies, which is not suitable for time-critical jobs. Nevertheless, the cloud environment is well-suited for jobs that can tolerate latencies.

4 OFFLOADING ORCHESTRATOR

The Offloading Orchestrator (OO) operates at an edge server and has five main functions: receive reservation requests from devices, assign server execution time to requests via scheduling optimization, receive accepted jobs from devices, assign jobs to proper processing units according to the schedule, and monitor the status of the servers on the network as shown in Fig. 2. Reservation requests from devices are placed in a reservation queue that the OO server periodically reviews to send the accumulated requests to the job scheduler. Then, the job scheduler component of the OO server runs an optimization algorithm called Real-time Priority-aware Semi-greedy (RT-PSG) to schedule job reservation requests to servers.

Details of the RT-PSG algorithm are provided in Section 5. Finally, the resource monitoring component of the OO server constantly scans the status of the other edge servers and the cloud generated by a management service running on each server. This monitoring component is able to report the status of the edge-cloud servers before each scheduling cycle.

4.1 Timing model

The response time of a job x^i is determined by 1) the transmission time from the OO to the edge server v, $T^{tx}_{x,OO\to v}$, 2) the waiting time at edge server v while higher priority jobs are executed, T^{w}_{v}, 3) the execution time at edge server v, $T^{exe}_{x^i,v}$, and 4) the reverse path

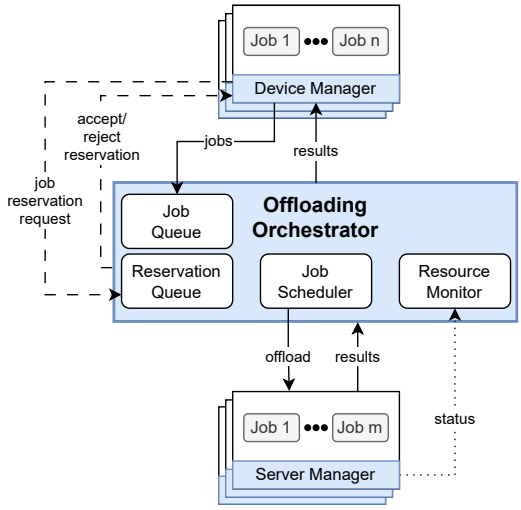


Figure 2: Offloading Orchestrator architecture.

of transmission of results back to the OO, $T^{tx}_{x,OO\leftarrow v}$. Note that a job that has a reservation does not need to be in a queue at the OO.

The transmission time from source to destination defined as

$$T^{tx}_{x^i,src\to dst} = \frac{\text{data size}}{b_{src\to dst}}, \tag{2}$$

where the data size is either in_{x^i} or out_{x^i} depending on the case and $b_{src\to dst}$ is the bandwidth of the link between the source and destination.

The waiting time at edge server v while previously scheduled jobs are executed is defined by

$$T^{w}_{v} = \sum_{y^i \in Y_v} T^{exe}_{y^i,v} \tag{3}$$

where Y_v is the set of previously scheduled jobs on server v.

Let s^{up}_{v} be the speedup of edge server v with respect to the local device. Then, the processing time of job x^i at edge server v is defined as

$$T^{exe}_{x^i,v} = \frac{w_{x^i}}{s^{up}_{v}}. \tag{4}$$

Therefore, the offloading time of job x^i when offloaded to edge server v is given by

$$T^{off}_{x^i,v} = T^{tx}_{x,OO\to v} + T^{w}_{v} + T^{exe}_{x^i,v} + T^{tx}_{x,OO\leftarrow v} \tag{5}$$

Note that the edge server running the OO can also execute offloaded jobs, in which case $T^{tx}_{x,OO\to v} = T^{tx}_{x,OO\leftarrow v} = 0$.

4.2 Energy Model

The energy model follows the model proposed in [1]. We only present the edge energy model, in which we have two components of the server being active and idle during execution of the edge servers. The edge energy consumption is shown in Eq.(6). The detailed calculation can be found in [1].

$$E_{Sys} = E^{act}_{Sys} + E^{idle}_{Sys}, \tag{6}$$

4.3 Resource allocation optimization

Let $\mathcal{J}$ be the set of all job reservation requests submitted to the reservation queue in OO. Let $\mathcal{T}$ be the time slots available at each scheduling cycle. The problem consists of mapping the set of jobs $\mathcal{J}$ to the set of edge-cloud servers V, and into a specific time slot, $\mathcal{S} : \mathcal{J} \rightarrow (V, \mathcal{T})$, such that each server runs only one job at a time, while minimizing edge energy consumption, E_{Sys}, and the time violation of the system, T^{viol}. Moreover, let T^{viol} be the total time violation in the system:

$$T^{viol} = \sum_{(x^i, v, t_m) \in \mathcal{S}} \max(ps_{x^i} + T^{off}_{x^i, v} - d_{x^i}, 0) \tag{7}$$

Formally, this multi-tier offloading reservation scheduling problem can be expressed as:

$$\begin{aligned} & \min E_{Sys}, \min T^{viol} \\ \text{subject to } & C1 : (x^i, v, t_m) \in \mathcal{S} \text{ s.t. } (x^i, w, t_l) \notin \mathcal{S}, \\ & \forall w \in V w \neq v, \forall t_l \neq t_m \in \mathcal{T} \\ & C2 : T^{off}_{x^i, v} \leq d_x, \forall x^i \in \mathcal{J}_h, \end{aligned} \tag{8}$$

where $\mathcal{J}_h$ is the set of jobs with hard deadline. Constraint $C1$ ensures that a job is only scheduled to strictly one combination of server and time slot during a scheduling cycle, and constraint $C2$ requires that each job with hard deadline meets its deadline. If a job cannot be scheduled to the system due to missed deadlines or a lack of free slots on all servers, the job is rejected.

To solve this optimization problem efficiently, we propose a low-complexity heuristic algorithms, which is described in the next section.

5 PROPOSED ALGORITHM: RT-PSG (REAL-TIME PRIORITY-AWARE SEMI-GREEDY)

In this section, we propose a heuristic algorithm, which is called RT-PSG, to schedule real-time job reservations that have been submitted by devices to the OO. RT-PSG extends the algorithms introduced by Azizi et al. [1] in which a semi-greedy approach was used for job scheduling in a fog environment to minimize energy consumption while meeting job deadlines. However, the problem defined in this paper is different from the one described in [1] as this work intends to schedule job reservations with multiple start times while in [1] jobs have the same start time that needs to be scheduled as soon as possible. Scheduling jobs with multiple start times requires not only the mapping of jobs to servers, but also a time slot assignment according to each jobs start time and deadline.

The RT-PSG semi-greedy approach prioritizes job reservation requests first by deadline type and then by urgency. Priority by deadline type is defined to give more priority to requests with hard deadlines over soft deadlines. The reason for this is that jobs with soft deadlines can tolerate delayed results, while jobs with hard deadlines will fail. The priority by urgency is given to jobs that have an earlier start time. Then, RT-PSG goes through the prioritized list of job reservations and successively assigns to a job the best server and time-slot available at the moment. The best server is determined by estimating the response time, $T^{off}_{x^i, v}$, the edge energy consumption, E_{Sys} , and the time violation, T^{viol}. Greedy approaches select the best possible result at the current decision point, while RT-PSG is semi-greedy because it includes a randomized step in the server selection. Our proposed RT-PSG consists of the following steps:

Step 1: Jobs are sorted by deadline type and by urgency in ascending order.

Step 2: Find a suitable time slot to start the job on each server that doesn't conflict with any other higher priority job previously scheduled to the servers.

Step 3: Calculate the response time of the job on each server using Eq. (5). Add all servers with response time to *ValidSList*. Add the servers that can meet the job deadline to the list of deadline satisfying servers, *DSList*.

Step 4: If *DSList* is not empty, such that at least one edge/cloud server can meet the job deadline, estimate the energy consumption of the system (Eq. 6) for each server up until this point where the current job is tentatively scheduled to such server. Add the least energy consuming servers to the Restricted Candidate List (RCL). Finally, randomly choose one server from RCL to schedule the job at the time slot previously identified.

Step 5: If *DSList* is empty, i.e. none of the edge/cloud servers can meet the specified job deadline, check if the job has a hard or a soft deadline.

Step 6: If *DSList* is empty and the job has a hard deadline, reject the job.

Step 7: If *DSList* is empty and the job has a soft deadline, select the server with the shortest response time in *ValidSList*, i.e., the one that violates the deadline the least. If *ValueSList* is empty, that is, all servers are full and no one is able to fit this job, reject the job.

The algorithm 1 demonstrates the process of finding a free time slot in a specific server in **Step 2**. First, we iterate over the list of jobs assigned to each server in search for one free slot per server that meets the requirements of the new job. A slot is found in a server when it is sufficiently large to accommodate the new job and it is free within the job's start time and the job's deadline. When a free slot is found in a server, the job is added to a copy of the current list of jobs assigned to the server which represents one scheduling option for the job. When the FIND-FREE-SLOT algorithm finds many scheduling options, i.e., many servers with a free slot, the optimized server selection process is handled by the succeeding algorithms. When none of the servers has a free slot, the job reservation is rejected.

Apart from that, other Steps are performed similar to [1], including the semi-greedy strategy to select the most energy saving servers, and the strategy to select the least time violating server when there is no server that can meet a soft deadline.

6 SIMULATION CONFIGURATION AND RESULTS

The proposed RT-PSG algorithm has been evaluated using a simulated scenario in which a set of job reservation requests arrive to a centralized entity, named OO, connected to a network of cloud and edge servers. In this section, we describe the simulation configuration and present the results.

Algorithm 1 FIND-FREE-SLOT

Input: $x^i, s', T^{exe}_{x^i,s'}$
Output: updated s' jobs with x^i in a suitable t

$dl \leftarrow d_{x^i}$ **if** x^i has hard deadline **else** $tout_{x^i}$
$init \leftarrow ps_{x^i}$
$index \leftarrow NULL$
if $s'.jobs = \varnothing$ **then**
  $index \leftarrow 0$
else
  for each job y^i in $s'.jobs$ **do**
    if $y^i.jobEnd \leq init$ **then**
      skip iteration
    else if $y^i.jobInit - init \geq T^{exe}_{x^i,s'}$ **then**
      $index \leftarrow y^i.index$
      break
    else if $y^i.jobEnd \geq dl$ **then**
      break
    end if
    $init \leftarrow y^i.jobEnd$
  end for
  if $index = NULL$ **then**
    if $0 < dl - s'.jobs.last.jobEnd \leq T^{exe}_{x^i,s'}$ **then**
      $index \leftarrow s'.jobs.last.index + 1$
      $init = max(s'.jobs.last.jobEnd, ps_{x^i})$
    end if
  end if
end if
if $index \neq NULL$ and $init + T^{exe}_{x^i,s'} \leq dl$ **then**
  $x^i.jobInit \leftarrow init$
  $x^i.jobEnd \leftarrow init + T^{exe}_{x^i,s'}$
  $s'.jobs \leftarrow s'.jobs[:index] \cup [x^i] \cup s'.jobs[index:]$
end if

6.1 Simulation Configuration

To simulate the scenario, we need two datasets: one of realistic job reservation requests and another one of realistic edge and cloud server characteristics.

To build the jobs dataset, a periodic tasks dataset is created first to represent offloading devices. This tasks dataset closely follows the task configuration of [1], with extra values regarding the periodicity of tasks needed in our case. The tasks are of two catagories; One is light-weight hard real-time tasks (worst-case execution time 100-300 ms, deadlines 100-400 ms) and other one is resource-intensive soft real-time tasks (worst-case execution time 400-800 ms, soft deadlines 400-1100 ms). Input size for each task is between 100 Kb - 10 Mb and output size is between 1 Kb - 1 Mb. Task periods 1-3 s.

The server dataset consists of Edge and Cloud Servers. The edge servers have a speedup over local processing of 1.5 to 3 times, whereas the cloud servers have a speedup over local processing of 5 to 10 times. The power consumption of the edge and cloud servers can be found in [1]. To stress the servers in this scenario, the number of jobs varies between 20 and 100. Half of the jobs have hard deadlines, while the other half have soft deadlines. During the simulation, the number of edge and cloud servers are fixed to 3 and 2 respectively, in accordance to an industrial scenario where devices can reserve time in nearby edge servers through the OO.

The simulation scripts are written in Python 3.10 and executed on an Intel Xeon CPU E5-2680, with 2.7GHz clock speed, 8 cores, 6 Gb of RAM, and over a Ubuntu 22.04 operating system.

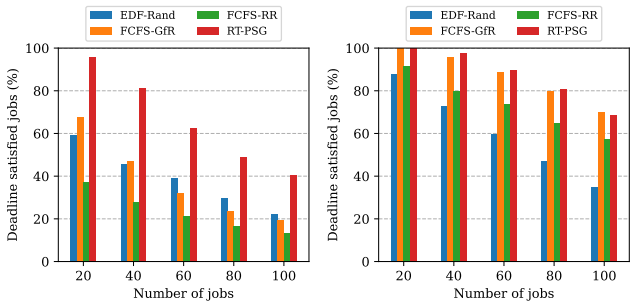
6.2 Baseline algorithms

To compare the performance of RT-PSG, we implement the following three baseline algorithms:

- **Earliest Deadline First - Random (EDF-Rand)**: This algorithm sorts the jobs to be scheduled by their deadline in ascending order in Earliest Deadline First (EDF) fashion, then randomly assigns a server to it (Rand).
- **First Come First Served - Greedy for Response Time (FCFS-GfR)**: This algorithm traverses the list of jobs as they arrives in First Come First Served (FCFS) fashion and sequentially assigns to it the server with the fastest response time in a Greedy for Response Time (GfR) strategy.
- **First Come First Served - Round Robin (FCFS-RR)**: This algorithm traverses the list of jobs as they arrive in FCFS fashion and rotates the server assignment in a circular fashion among the available servers in a Round Robin (RR) strategy.

6.3 Results

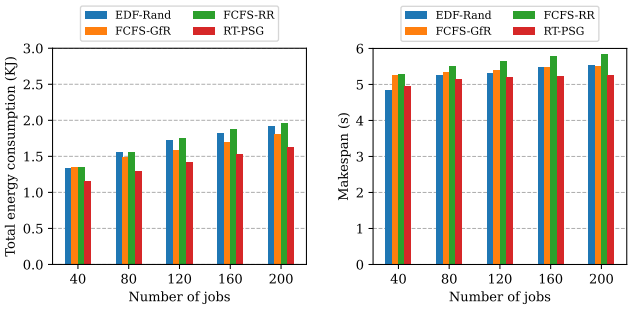
In this section, the performance of the RT-PSG algorithm for server time reservation with the datasets and evaluation scenarios are presented. The RT-PSG algorithm has been compared with 3 baseline algorithms: EDF-Rand, FCFS-RR and FCFS-GfR to evaluate its performance.



(a) Hard deadline. **(b) Soft deadline.**

Figure 3: Percentage of deadline satisfied jobs.

Figure 3 displays deadline satisfied jobs, while Figure 3a focuses on algorithms' performance with hard deadlines. Among them, RT-PSG excels in maintaining high deadline satisfaction, significantly outperforming FCFS-GfR and EDF-Rand, as other algorithms don't distinguish between hard and soft deadlines, leading to reservations that miss hard deadlines. On the contrary, RT-PSG rejects unsatisfiable hard deadline jobs, allowing space for others. On the other hand, FCFS-RR prioritizes fair workload distribution over deadlines, resulting in the lowest hard deadline satisfaction, consistently below 40%.



(a) Energy consumption of edge servers.

(b) Makespan.

Figure 4: Energy consumption of edge servers and system makespan.

Figure 3b shows how the different algorithms handle reservations for jobs with soft deadlines. Noticeably, even when RT-PSG gives priority to hard deadline jobs, it is also able to satisfy soft deadline jobs by the level of FCFS-GfR, which does not consider the hardness of job deadlines. FCFS-GfR tends to be superior to RT-PSG when workload increases; however, this level of satisfaction of deadlines comes at the cost of missing a considerable amount of hard deadlines in the same reservation cycle. Regarding the performance of the EDF-Rand algorithm, its random strategy to select servers makes it choose busy servers more often than the other algorithms, which does not help with hard or soft deadline satisfaction.

Figure 4a presents the energy consumed by edge servers when following a schedule, as this is one of the optimization goals of RT-PSG. In this scenario, RT-PSG is able to save the most edge energy not only due to the early rejection of hard deadline jobs, but also due to its randomized selection of servers among the top performers, which enables it to choose cloud servers more often compared to FCFS-GfR, which tends to focus on edge servers due to their fast response time. This is corroborated in Figure 4b, which estimates the makespan of the system including edge and cloud servers and shows how the RT-PSG schedule would be executed faster.

Regarding EDF-Rand and FCFS-RR, have a similar level of edge server energy consumption for different number of jobs in the test. Since none of these two algorithms take the server response time in to account make their scheduling decisions, both are more likely to schedule jobs in unsuitable servers than their counterparts. In this scenario, RT-PSG offers a more balanced trade-off between deadline misses and edge server energy consumption by achieving a high rate of deadline satisfaction while keeping a low edge server energy consumption.

7 CONCLUSIONS

This work presented a centralized server reservation system for real-time applications that offload periodic task instances (jobs) to the edge/cloud. A semi-greedy heuristic algorithm, RT-PSG, is proposed to optimize reservation scheduling so that it prioritizes hard deadline requests over soft deadline requests and produces a schedule that minimizes the energy consumption of edge servers. RT-PSG was evaluated against three baseline algorithms and it was found that the prioritization and server selection strategies of this algorithm can balance the trade-off between energy consumption and the percentage of deadline satisfied jobs. The results showed that RT-PSG achieves higher percentages of jobs with met deadline, while energy consumption is lower. In the future, we plan to compare the proposed algorithms with other related studies. Additionally, we can extend the simulation configurations and scenarios to consider more realistic setups. Moreover, we can consider monetary costs and how we can optimize our task scheduling based on that, and complexity analysis can be added to show how our proposed algorithm works compared to other algorithms.

ACKNOWLEDGMENTS

This work was partially supported by the project AORTA (Advanced Offloading for Real-Time Applications) that has received funding from Swedish Innovation Agency (VINNOVA) under grant agreement No 2022-03039.

REFERENCES

[1] Sadoon Azizi, Mohammad Shojafar, Jemal Abawajy, and Rajkumar Buyya. 2022. Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach. *Journal of network and computer applications* 201 (2022), 103333.

[2] Mir Salim Ul Islam, Ashok Kumar, and Yu-Chen Hu. 2021. Context-aware scheduling in Fog computing: A survey, taxonomy, challenges and future directions. *Journal of Network and Computer Applications* 180 (2021), 103008.

[3] Congfeng Jiang, Xiaolan Cheng, Honghao Gao, Xin Zhou, and Jian Wan. 2019. Toward computation offloading in edge computing: A survey. *IEEE Access* 7 (2019), 131543–131558.

[4] Taehoon Kim, Yongjae Kim, Emmanuella Adu, and Inkyu Bang. 2023. On Offloading Decision for Mobile Edge Computing Systems Considering Access Reservation Protocol. *IEEE Access* (2023).

[5] Hai Lin, Sherali Zeadally, Zhihong Chen, Houda Labiod, and Lusheng Wang. 2020. A survey on computation offloading modeling for edge computing. *Journal of Network and Computer Applications* 169 (2020), 102781.

[6] Pavel Mach and Zdenek Becvar. 2017. Mobile edge computing: A survey on architecture and computation offloading. *IEEE communications surveys & tutorials* 19, 3 (2017), 1628–1656.

[7] Aamir Mahmood, Luca Beltramelli, Sarder Fakhrul Abedin, Shah Zeb, Nishat I Mowla, Syed Ali Hassan, Emiliano Sisinni, and Mikael Gidlund. 2021. Industrial IoT in 5G-and-beyond networks: Vision, architecture, and design trends. *IEEE Transactions on Industrial Informatics* 18, 6 (2021), 4122–4137.

[8] Sudip Misra and Niloy Saha. 2019. Detour: Dynamic task offloading in software-defined fog for IoT applications. *IEEE Journal on Selected Areas in Communications* 37, 5 (2019), 1159–1166.

[9] Saad Mubeen, Pavlos Nikolaidis, Alma Didic, Hongyu Pei-Breivold, Kristian Sandström, and Moris Behnam. 2017. Delay Mitigation in Offloaded Cloud Controllers in Industrial IoT. *IEEE Access* 5 (2017), 4418–4430.

[10] Anas Toma and Jian-Jia Chen. 2013. Server resource reservations for computation offloading in real-time embedded systems. In *The 11th IEEE Symposium on Embedded Systems for Real-time Multimedia*. IEEE, 31–39.

[11] Peng Yang, Ning Zhang, Yuanguo Bi, Li Yu, and Xuemin Sherman Shen. 2017. Catalyzing cloud-fog interoperation in 5G wireless networks: An SDN approach. *IEEE Network* 31, 5 (2017), 14–20.

[12] Jianhui Zhang, Jiacheng Wang, Zhongyin Yuan, Wanqing Zhang, and Liming Liu. 2023. Offloading Demand Prediction-driven Latency-aware Resource Reservation in Edge Networks. *IEEE Internet of Things Journal* (2023).