Optimal Proactive Resource Allocation at the Extreme Edge

Rawan F. El Khatib[1], Sara A. Elsayed[2], Nizar Zorba[3], and Hossam S. Hassanein[2]

[1]Department of Electrical and Computer Engineering, Queen's University, Canada
[2]School of Computing, Queen's University, Canada
[3]Department of Electrical Engineering, Qatar University, Qatar
rawan.elkhatib@queensu.ca, selsayed@cs.queensu.ca, nizarz@qu.edu.qa, hossam@cs.queensu.ca

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9838897

Abstract—**Edge Computing (EC) has emerged as a key enabling paradigm for latency-critical and/or data-intensive applications. Recently, recycling abundant yet underutilized computational resources of the Extreme Edge Devices (EEDs), such as smartphones, laptops, connected vehicles, etc, has been explored. This is since EEDs can bring the computation service much closer to the edge, which can drastically reduce the delay. However, resource allocation in such environments typically follows a reactive approach, which can lead to increased delay and wasted resources. In this paper, we introduce the Optimal Proactive Resource Allocation (OPRA) benchmark to quantify the potential gains of proactive resource allocation in EC environments. OPRA exploits the predictability of request patterns to proactively perform resource allocation and create compute clusters that take future task and resource dynamics into consideration. Specifically, OPRA formulates the resource allocation problem as a Binary Integer Linear Program (BILP) problem, where it aims to minimize the total delay under full task assignment and computation capacity constraints. The optimal solution acquired under perfect knowledge acts as the upper bound on the achievable potential of predictive proactive resource allocation schemes. The effect of erroneous predictions on the performance of OPRA is also investigated. Extensive simulation results show that OPRA outperforms a reactive baseline by yielding a 50% decrease in the subtask dropping rate and 97% decrease in the service capacity.**

I. Introduction

With the advent of the Internet-of-Things (IoT), it is anticipated that 125 billion IoT devices, such as smartphones, connected vehicles, laptops, etc., will be connected to the Internet by 2030 [1]. This rise contributes to the growing popularity of a broad range of latency-sensitive and/or data-intensive applications, such as healthcare, virtual reality, smart cities, etc. Cloud computing struggles to meet the strict Quality of Service (QoS) requirements of such applications. This is since data must be fully transmitted to distant data centers for processing. To respond to these soaring demands, Edge Computing (EC) has emerged as a key enabling paradigm, expected to exhibit a Compound Annual Growth Rate (CAGR) of 38.4% by 2028 [2]. The core idea of EC is to push data processing closer to the edge where data is generated, which drastically reduces the communication latencies and alleviates the excessive traffic load at back-haul links [3].

In EC, performance gain is largely dependent on efficient computation offloading decisions. The majority of developed EC platforms rely on dedicated compute-capable edge servers to carry the offloaded computational tasks [3]. Recently, there has been a growing body of literature [4]–[6] that proposes to capitalize on the proliferation of IoT devices, also referred to as Extreme Edge Devices (EEDs) [7], and their increasingly powerful processing capacities within the EC paradigm. In this EED-enhanced model of EC, the underutilized computational resources of EEDs are recruited to augment the resource pool and improve the offloading service.

Computation offloading services that existing resource allocation schemes deal with are classified into two categories: a) binary (full) offloading, where a computational task is fully delegated to another EED for execution, and b) partial offloading, where a task is partitioned into smaller subtasks for execution at a group of collaborating EEDs [8]. In this article, we are concerned with the latter category, which is prevalent in scenarios where a task is too large to execute at a single EED, or when different portions of the task become ready for processing at different time instants. For such scenarios, most existing works regard the partitioned subtasks as independent [9], [10]. This implies that a) the need to ensure that all subtasks are assigned is overlooked, and b) the effect of subtask execution on the parent task successful completion is ignored. In this work, we deal with partial offloading where the successful completion of any task depends on the execution of all of its subtasks.

To illustrate the significance of the aforementioned notion, consider an example of a delay-sensitive data-intensive video processing task. The task aims to count the number of people entering a venue (e.g., a shopping center), to ensure that COVID-19 capacity restrictions are not violated. Several CCTV cameras capture crowds entering from various points of entry. All CCTV cameras are connected to a control room where video recordings are partitioned into smaller segments to be offloaded to multiple nearby EEDs, such as crowd members' smartphones, tablets, parked vehicles, etc. Because the video recording is partitioned as it is taken, the segments become available for processing at different times. That is to say, the subtasks have different arrival times. Furthermore, the workers (i.e., EEDs) have heterogeneous spatio-temporal resource availability, indicating that a subtask will incur different execution and communication delays at different EEDs. Evidently, the successful completion of the video processing task is contingent on the completion of all subtasks. In other words, if all segments return a crowd count except for a single failing subtask, the parent task is considered incomplete. Existing works disregard such unity and offer no guarantee that all subtasks will be executed, which lowers the rate of successful task completion. In addition, most existing resource allocation schemes adopt a reactive approach, where resource allocation decisions are made in response to incoming requests after their arrival. As a result, resource allocation lags far behind resource requests, leading to imbalanced resource utilization and wasted resources.

To address the aforementioned challenges, we introduce the Optimal Proactive Resource Allocation (OPRA) benchmark. OPRA strives to proactively allocate partitioned subtasks belonging to a single parent task to a group of collaborating EEDs that form a compute cluster. The objective is to form resource-sufficient compute clusters for each parent task, such that all subtasks are executed to achieve minimal latencies. Furthermore, we address the sporadic nature (i.e., heterogeneous arrival times) of subtasks by aiming to exploit the predictability of task dynamics (e.g., prediction of venue peak hours). Leveraging this predictability allows to *proactively* create subtask-resource mappings in lieu of waiting for task requests to arrive. This proactive resource allocation approach can track, learn and predict task dynamics ahead of time, and hence offers more flexibility to make highly informed resource allocation decisions.

To the best of our knowledge, OPRA is the first to explore proactive resource allocation in the context of EED-enhanced EC. In addition, OPRA determines the upper bound on the achievable potential of predictive resource allocation. This is done by formulating the resource allocation problem as a Binary Integer Linear Program (BILP) under perfect knowledge (i.e., no prediction errors) of task arrival times and workers' spatio-temporal availability. This allows to quantify the potential gains of resource allocation schemes and evaluate the performance gap in comparison with the optimal solution. Additionally, we study the realistic scenario in which estimates of task arrival times can be erroneous. We refer to the latter variation as Robust Hybrid Resource Allocation (RHRA). The remainder of the paper is organized as follows. In section II, we discuss some related work. Section III introduces the proposed benchmark (OPRA). Section IV presents RHRA. Section V discusses the performance evaluation and simulation results. Finally, section VI presents our conclusions and future work directions.

II. Related Work

The potential of augmenting EC computational capacity with EEDs-contributed resources has been explored in the literature recently. This approach is shown to bring many benefits, such as natural proximity to the edge, lower latencies, fault tolerance, better scalability, and improved cost-effectiveness [4], [5]. This model of EC has been referred to as device-enhanced EC [8], collaborative multidevice computing [6], opportunistic EC [4], and mobile ad hoc computing [3]. These works take different perspectives on the type and level of interaction and collaboration with edge servers.

Most existing resource allocation schemes for EED-enhanced EC follow a reactive approach, where the scheduling entity responds to tasks after they are initiated by their requesters. Most works aim to allocate computational resources such that the latency and/or the energy consumption is minimized through the optimization of communication and computation resources [8]. The work in [9] minimizes the latency by optimizing task assignment jointly with time and power allocation under individual energy constraints, whereas [10] minimizes the latency by jointly considering the tasks assignment, the task offloading times and rates, the local and remote task execution times and computation frequencies, and the results downloading times and rates. In both these works, there exists a single requester with multiple independent tasks with identical arrival times. On the other hand, the work in [11] considers a system with multiple requesters, each with a single task to execute locally or by a worker, with the objective of minimizing the overall energy consumption.

The works in [12] and [13] also consider a system with multiple requesters each with a single task, with the objective of jointly minimizing latency and energy consumption. A more recent work is introduced in [14], where the authors focus on fault-tolerant decentralized formation of compute clusters, such that EEDs collaborate to execute latency-sensitive tasks. However, these schemes cannot cater to critical scenarios where a task's completion is contingent on the execution of a batch of subtasks. Moreover, these schemes cannot accommodate the sporadic arrival times efficiently, as they all employ a reactive resource allocation approach. To the best of our knowledge, our work is the first to employ a proactive resource allocation approach that relies on the predictability of task arrival times and spatio-temporal distribution of workers.

III. Optimal Proactive Resource Allocation (OPRA)

In this section, we present our assumptions, the system model, the problem formulation, and the functionality of our OPRA scheme.

A. Assumptions and System Model

Our system consists of a centralized entity, a set of requesters and a set of workers (i.e., EEDs). The centralized entity, referred to as the scheduler, acts as a mediator between requesters and workers. The scheduler is responsible for probing computational resources from heterogeneous EEDs and creating appropriate subtask-resource mappings. The scheduler is hosted at an edge server that is able to communicate with both requesters and workers. It is noteworthy that our scheme does not consider the possibility to offload computational tasks to the edge server. In other words, the computational resources considered in this article are exclusively those offered by EEDs.

The scheduler collects information related to workers' location, duration of availability, and amount of dedicated resources. This is done in exchange for incentives provided to the recruited EEDs and based on a privacy agreement. Moreover, we assume that the scheduler has a prediction module that enables it to perform accurate predictions of future task dynamics. Predictors such as in [15] can be utilized for this purpose. These predictions offer insights into the tasks (and subtasks) arrival times and workload characterization (e.g., amount of required resources, execution time, etc.). In this manner, our scheduler attains full knowledge of both the spatio-temporal availability of the resources and tasks dynamics. This future knowledge embodies the core enabler of our proactive resource allocation scheme.

Without loss of generality, we assume that this knowledge is acquired by the scheduler for the system's sojourn time, during which the characteristics of workers and tasks are invariant. Hence, time is divided into sequential execution sessions $\{s_1, s_2, s_3, ...\}$, where the length of each session is equal to the sojourn time. Ahead of the start of each session, the scheduler proactively creates subtask-resource mappings in accordance to our problem formulation, as explained in the sequel. The actual length of the sessions (i.e., sojourn times) depends on the scenario in question. For example, in a scenario where computational resources are offered by vehicles waiting at a traffic light, the sojourn time is in the scale of a few minutes (e.g., 5 minutes). On the other hand, in our shopping center scenario mentioned earlier, the sojourn time is expected to be larger (e.g., 30 minutes) as crowd members are predicted to spend more time near the scheduler. For simplicity, we abandon the session notation hereafter, emphasizing that all variables introduced in the following are invariant during an individual execution session.

Given the above assumptions, consider a set of $\mathcal{I}$ requesters each with a task, where each task i comprises J_i subtasks denoted by set $\mathcal{J}_i$. Each subtask $\langle i,j\rangle$ is characterized by the subtask arrival time $t_{i,j}^{arrival}$, the required computational intensity $q_{i,j}$ in CPU cycles/bit, designating the number of CPU cycles required for computing one input bit, the input subtask size $L_{i,j}^{in}$ and the output subtask size $L_{i,j}^{out}$ in bits. Let $\mathcal{K}$ denote the set of available workers, where each worker's maximum computational capacity is Z_k in CPU cycles/second, and δ_k denotes the percentage of occupied processing capacity due to local tasks. Hence, each worker has a maximum available CPU frequency $C_k^{max} = (1-\delta_k)Z_k$. When subtask $\langle i,j\rangle$ is assigned to worker k, it is allocated a CPU frequency $c_{i,j}^k = u_{i,j}^k C_k^{max}$, where $u_{i,j}^k \in (0,1]$ is the available rate of worker k computational resources, set in accordance to the incentives offered by the requester for subtask $\langle i,j\rangle$. The distance between the requester i and worker k is d_i^k, the upload/download data rate is R_i^k, and the propagation speed is v. Finally, for each worker-subtask pair, we define the interval $A_{i,j}^k$ which spans the start time of the subtask $\langle i,j\rangle$ until its completion at worker k.

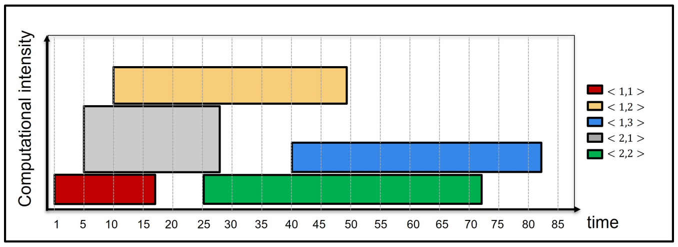


Fig. 1: An example of subtask assignments at a worker k. Each rectangle designates a different subtask $\langle i,j\rangle$. The length of each rectangle represents the execution time, and the width represents the required computational intensity. The arrival time of each subtask is shown on the time axis in seconds.

B. Problem formulation

Our objective is to minimize the total delay in the system by proactively forming a compute cluster for each task such that the corresponding resources of the cluster are available at the subtasks arrival time and incurs minimal total delay. The total delay D_{total} is the sum of incurred delay for each subtask-resource mapping decision, denoted by $D_{i,j}^k$. For subtask $\langle i,j\rangle$ at worker k, $D_{i,j}^k$ is expressed as follows:

$$D_{i,j}^k = \frac{q_{i,j} L_{i,j}^{in}}{c_{i,j}^k} + 2 \times \frac{d_i^k}{v} + \frac{L_{i,j}^{in} + L_{i,j}^{out}}{R_i^k} \tag{1}$$

The three terms represent the execution delay, the round trip propagation delay, and the transmission delay, respectively. Accordingly, the problem is formulated as a Binary Integer Linear Program (BILP), where the decision variable $x_{i,j}^k$ is set to 1 if subtask $\langle i,j\rangle$ is assigned to worker k at time interval $A_{i,j}^k$, and 0 otherwise. The mathematical formulation of the said objective is shown below.

$$\underset{x_{i,j}^k,\,\sigma_i}{\text{minimize}} \quad \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}_i} \sum_{k\in\mathcal{K}} D_{i,j}^k x_{i,j}^k$$

subject to

$$\text{C1: } \sum_{k\in\mathcal{K}} x_{i,j}^k \le 1 \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i,$$

$$\text{C2: } \sum_{\langle i,j\rangle\in\psi_n} c_{i,j}^k x_{i,j}^k \le C_k^{max} \qquad \forall \psi_n \in O_k, \forall k \in \mathcal{K},$$

$$\text{C3: } \sum_{i\in\mathcal{I}} \sigma_i = \alpha \quad ,$$

$$\text{C4: } \sum_{j\in\mathcal{J}_i} \sum_{k\in\mathcal{K}} x_{i,j}^k \ge J_i[\sigma_i(M+1) - M] \qquad \forall i \in \mathcal{I},$$

$$\text{C5: } \sum_{j\in\mathcal{J}_i} \sum_{k\in\mathcal{K}} x_{i,j}^k < J_i[\sigma_i(M+1) + 1] \qquad \forall i \in \mathcal{I}$$

Constraint C1 indicates that each subtask is assigned to at most one worker. Constraint C2 ensures that the total assignments to each worker do not exceed its computational capacity C_k^{max} at any overlapping time interval. To ensure this condition, we define O_k as a set of sets ψ_n, $n \in \mathbb{N}$, where ψ_n is a set of all overlapping intervals at worker k. Then, for all the subtasks in the set ψ_n, the sum of the allocated CPU frequencies should not exceed C_k^{max}. In Fig. 1, we show an example of sporadic subtasks at an arbitrary worker k. In this case, $\psi_1 = \{\langle 1,1\rangle \langle 1,2\rangle \langle 2,1\rangle\}$, $\psi_2 = \{\langle 1,2\rangle \langle 2,1\rangle \langle 2,2\rangle \}$, and $\psi_3 = \{\langle 1,2\rangle \langle 1,3\rangle \langle 2,2\rangle \}$, with $O_k = \{\psi_1, \psi_2, \psi_3\}$.

To impose the condition on full task execution by assigning all subtasks, we define the service capacity metric as the number of fully assigned tasks. We impose constraint C3 to ensure that the service capacity is maintained above a predefined threshold α, where $\alpha \in \mathbb{N}$. For this purpose, we define the indicator variable σ_i, which is set to 1 if all J_i subtasks of task i are assigned, and 0 otherwise. This is to say, for any task i, $\sigma_i = 1$ if $\sum_{j\in\mathcal{J}_i}\sum_{k\in\mathcal{K}} x_{i,j}^k = J_i$, and $\sigma_i = 0$ if $\sum_{j\in\mathcal{J}_i}\sum_{k\in\mathcal{K}} x_{i,j}^k < J_i$. To incorporate this conditional statement in the constraints, we introduce the additional constraints C4 and C5, in which the variable M is set to a large positive value ($M >> \sum_{j\in\mathcal{J}_i}\sum_{k\in\mathcal{K}} x_{i,j}^k$). C4 and C5 are verified as follows: substituting $\sigma_i = 0$ in both constraints will result in the inequality $-J_iM \le \sum_{j\in\mathcal{J}_i}\sum_{k\in\mathcal{K}} x_{i,j}^k < J_i$. Alternatively, substituting $\sigma_i = 1$ will result in the inequality $J_i \le \sum_{j\in\mathcal{J}_i}\sum_{k\in\mathcal{K}} x_{i,j}^k < J_i(M+2)$. Thus, C5 and C6 serve as artificial constraints designed to verify C3.

In summary, our OPRA scheme works as follows. Ahead of the start of an execution session, the scheduler exploits its prediction module to estimate subtask arrival times and workload characteristics. The scheduler also collects information from workers on their future locations and amounts of computational resources available for the next session. Then, the scheduler solves the optimization problem formulated above to minimize the total delay of all tasks (and subtasks), subject to service capacity and individual workload constraints. The result is subtask-resource mappings for the entire duration of the execution session, such that in total, α tasks are fully executed with minimal delay.

IV. Robust Hybrid Resource Allocation (RHRA)

In the previous section, we focused on the ideal scenario where tasks' characteristics in terms of arrival time and workload are perfectly predictable. In this manner, the scheduler possesses full certainty about the workload demands in the future session, and proactively creates subtask-resource mappings. Realistically speaking, prediction inaccuracies are almost unavoidable, and evoke the possibility of (sub)task failure and resource waste. In this section, we examine the scenario of imperfect knowledge due to inaccurate predictions of task characteristics. Specifically, we address the scenario in which errors can occur when predicting subtask arrival times, in order to isolate its effect on design and performance. Specifically, we consider prediction errors in which a subtask arrives after its predicted time.

For this purpose, we present the Robust Hybrid Resource Allocation (RHRA) scheme. The core idea of RHRA is that it captures the predictable and sustained resource demand patterns from historical data and proactively provisions computational resources for them. At runtime, the deviations between the actual and predicted resource demands are handled by reactive provisioning. In this manner, RHRA employs a hybrid proactive-reactive approach that aims to mitigate the effect of erroneous predictions of arrival times.

Algorithm 1 Robust Hybrid Resource Allocation (RHRA).

```
 1: solve OPRA.
 2: repeat
 3:     repeat
 4:         if prediction error in $\langle i,j\rangle$ then
 5:             insert $\langle i,j\rangle$ at the rear of $Q$
 6:         end if
 7:     until end of current time period
 8:     for all $\langle i,j\rangle \in Q$ do
 9:         get worker $\hat{k}$ with minimum $D_{i,j}^{\hat{k}}$ and $c_{i,j}^{\hat{k}} \le C_{\hat{k}}^{max}$
10:         if $\exists \hat{k}$ then
11:             assign $\langle i,j\rangle$ to $\hat{k}$
12:             update $D_{i,j}^{\hat{k}}$
13:         else
14:             drop $\langle i,j\rangle$ from $Q$
15:         end if
16:     end for
17: until end of execution session
```

RHRA operates under the same assumptions as OPRA, with the exception that it accommodates erroneous task predictions rather than assuming perfect knowledge. Specifically, before the beginning of an execution session, RHRA proactively produces optimal subtask-resource mappings by solving the problem formulated in the previous section based on the available predictions of arrival times. At runtime, the execution session is divided into equal time periods. During each period, the following procedure, illustrated in Algorithm 1, is performed. The scheduler monitors the status of subtasks, and those which deviate from their expected arrival times are held off and added to a virtual queue Q at the scheduler (line 5). Then, at the end of the time period, the scheduler invokes a recovery mechanism based on a greedy reactive approach. Specifically, for each subtask in Q, the scheduler looks for a worker $\hat{k}$ that achieves the minimum $D_{i,j}^k$ without exceeding its computational capacity. If such a worker exists, then subtask $\langle i,j\rangle$ is assigned to $\hat{k}$ (line 11), and the value $D_{i,j}^k$ is updated to reflect the excess delay from the erroneous prediction and the waiting time until the end of the time period (line 12). If none of the workers satisfy these conditions, the subtask is dropped (line 14). The intuition behind the reactive recovery mechanism is to mitigate the effects of erroneous predictions on the proactively created subtask-resource mappings by resorting to unused computation resources. In this manner, it is guaranteed that the disruptions caused by erroneous predictions are isolated from accurate ones, while still seeking to achieve the required service capacity with minimal total delay.

V. Performance evaluation

In this section, we evaluate the performance of OPRA and RHRA compared to a reactive resource allocation approach. The reactive baseline follows a greedy method, where the execution session is divided into equally spaced time periods. During each period, the scheduler collects arriving subtask requests. Then, at the end of each period, the scheduler reacts to the received subtask requests for execution in the beginning of the next time period. Subtask requests are handled in the order they arrive by sequentially assigning subtasks with the earliest finish time until all the available computational resources are exhausted.

We employ the following performance metrics: 1) the average total delay, which measures the sum of the time since a subtask arrives until it is fully executed and its result is returned to the requester for all the subtasks in the system, 2) the average subtask dropping rate, which is the number of dropped subtasks to the total number of subtasks in the system, and 3) the average service capacity ratio, which is the number of fully executed tasks to the total number of tasks in the system.

A. Simulation Environment

We consider an area of $200m \times 200m$ where workers and requesters are uniformly distributed. The number of workers is set to 10, the number of tasks is varied in $[5, 10, 15, \dots, 60]$, and the number of subtasks per task is set to 5. The execution session is set to 900 seconds (i.e., 15 minutes), and subtask arrival times are uniformly distributed in $\mathcal{U}\{1, 900\}$ seconds. The computational intensity of the subtasks is uniformly distributed over the range $[1000, 3000]$ cycles/bit, the input subtask size is uniformly distributed in the range $[400, 4000]$ Kbit, and output subtask size is set to $0.05 \times L_{i,j}^{in}$. The maximum computation capacity is uniformly distributed in $[1.5, 3]$ GHz, and the percentage of occupied capacity is in $[0.1, 0.9]$. The upload/download data rate between workers and requesters is uniform over the range $[1, 10]$ Mbps.

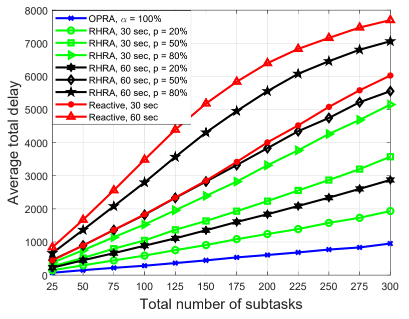
To study the effect of erroneous predictions, we vary the percentage p of subtasks for which the predictor module produces inaccurate arrival time estimations. The prediction error delay in RHRA is uniformly distributed in $[0, 10]$ seconds. For both the reactive and RHRA schemes, we show results when the time period is set to 30 and 60 seconds.

B. Simulation Results

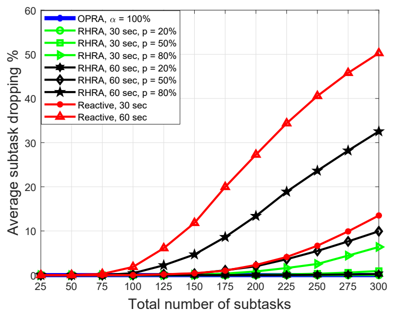
First, we evaluate the performance of OPRA, RHRA and the reactive baseline in terms of the average total delay in Fig. 2a. For OPRA, we fix the service capacity threshold α to 100% of the subtasks in the system, imposing a condition of full execution of all task requests. Since the average total delay accounts for the accumulative delay of all executed subtasks, all curves show an upward trend as the total number of subtasks increases. As depicted, OPRA provides the upper bound on the potential delay improvement. OPRA demonstrates a significant reduction in the average total delay compared to the reactive baseline in both scenarios for the time period length. As can be seen, the reactive baseline exhibits a greater increase in the average total delay compared to OPRA when the time period is set to 60 seconds. This is because arriving subtasks are stored and await until they are assigned in the next time period. The 30-second reactive baseline offers better performance at the expense of increased scheduling overhead. On the other hand, the performance gap between RHRA and OPRA grows larger as the percentage of inaccurate predictions and the time period length increase. This is because RHRA invokes the reactive greedy recovery mechanism for erroneous subtasks, capping the performance gain attained from the proactively created subtask-resource mappings. This degradation is further exacerbated when the time period is longer, as the total delay is further prolonged by the time an erroneous subtask is held in the virtual queue until execution. Hence, as can be seen, for high percentages of inaccurate predictions, RHRA approaches the performance of the reactive baseline for the same time period.

Second, we examine the average subtask dropping rate in Fig. 2b. As demonstrated in the Fig., OPRA provides the upper bound on the achievable subtask dropping rate, producing 0% subtask dropping rate for all simulated scenarios. This is because OPRA produces the optimal solution that adheres to the specified service capacity threshold α, which is set to 100% throughout the experiment. RHRA performs comparably when the percentage of inaccurate predictions is 20%, regardless of the time period, maintaining a subtask dropping rate below 1% as the total number of subtasks increases from 25 to 300. When the percentage of inaccurate predictions is 50%, RHRA sustains a subtask dropping rate below 1% and 10% compared to below 6% and 32% when the percentage of inaccurate predictions is 80%, for a time period of 30 and 60 seconds, respectively. This is because in a longer time period, there exists a higher possibility that erroneous subtasks will accumulate in the queue and compete for the same computational resources, causing the subtask dropping rate to rise. On the other hand, the reactive scheme exhibits a sharper increase in the subtask dropping rate as the total number of subtasks increases, peaking at 13% and 50% when the time period is 30 and 60 seconds, respectively. Thus, RHRA surpasses the equivalent time period reactive baseline even for high percentages of inaccurate predictions.

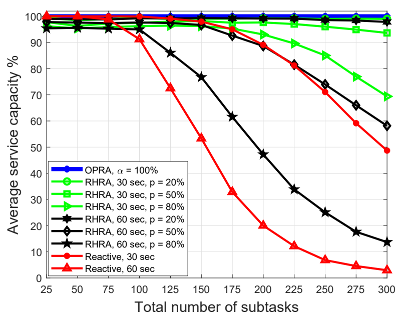
Finally, we investigate the performance of these schemes in terms the average service capacity. This is an important performance indicator, as it highlights the number of fully executed tasks, which cannot be clearly assessed from the subtask dropping rate. As shown in Fig. 2c, OPRA produces the upper bound on the achievable service capacity ratio, yielding 100% for all values of the total number of subtasks. When the percentage of inaccurate predictions is 20%, RHRA is on a par with OPRA, sustaining a service capacity rate above 97%. While the achieved service capacity declines as the percentage of inaccurate predictions, the time period length, and/or the total number of subtasks increase. This is due to the same reasons previously discussed. The reactive baseline yields high values of average service capacity when



(a) Average total delay



(b) Average subtask dropping rate



(c) Average service capacity rate

Fig. 2: Performance results over varying number of subtasks.

the total number of subtasks is low, before experiencing a sharp decline to 48% and 3% for 30- and 60-second time periods, respectively. This is because the reactive baseline does not prioritize the execution of subtasks belonging to tasks whose execution was already started, as opposed to OPRA and RHRA.

VI. Conclusions and Future Work

In this paper, we introduced OPRA, which exploits the predictability of task requests to proactively form resource-sufficient compute clusters under service capacity and computational capacity constraints. The problem was formulated as a BILP under perfect knowledge of task arrival times and workers' spatio-temporal availability. The optimal solution acts as the upper bound on the achievable potential of predictive proactive resource allocation in EED-enhanced EC environments, allowing to evaluate the performance gap and assess if there is room for improvement. In addition, the effects of erroneous predictions were investigated under RHRA. We conducted extensive experimental evaluations and showed that OPRA outperforms the reactive baseline by yielding a 50% decrease in the subtask dropping rate and 97% increase in the service capacity rate. In our future work, we aim to design a prediction module that poses an upper limit on the prediction error of subtask arrival times, and present a probabilistic heuristic solution to counteract the effects of prediction errors.

Acknowledgement

This research is supported by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant number ALLRP 549919-20. This research is also supported by Qatar University QUHI-CENG-21-22-1 project.

References

[1] K. Gyarmathy, “Comprehensive Guide to IoT Statistics You Need to Know in 2020,” March, 2021. Available from: https://www.vxchnge.com/blog/iot-statistics

[2] “Edge Computing Market Share & Trends Report,” Grand View Research, May, 2021. Available from: https://www.grandviewresearch.com/industry-analysis/edge-computing-market

[3] A. J. Ferrer, J. M. Marquéés, and J. Jorba, “Towards the Decentralised Cloud: Survey on Approaches and Challenges for Mobile, Ad Hoc, and Edge Computing,” in *ACM Computing Surveys,* vol. 51, no. 6, pp. 1-36, 2019, doi:10.1145/3243929.

[4] R. Olaniyan, O. Fadahunsi, M. Maheswaran, and M. F. Zhani, “Opportunistic Edge Computing: Concepts, Opportunities and Research Challenges,” in *Future Generation Computing Systems,* vol. 89, pp. 633–645, 2018. doi:10.1016/j.future.2018.07.040.

[5] Y. Sahni, J. Cao, S. Zhang and L. Yang, “Edge Mesh: A New Paradigm to Enable Distributed Intelligence in Internet of Things,” in *IEEE Access*, vol. 5, pp. 16441-16458, 2017, doi:10.1109/ACCESS.2017.2739804.

[6] W. Zhang, H. Flores and P. Hui, “Towards Collaborative Multi-device Computing,” in *Proc. IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops),* Greece, 2018, pp. 22-27, doi:10.1109/PERCOMW.2018.8480262.

[7] J. Portilla, G. Mujica, J. -S. Lee and T. Riesgo, “The Extreme Edge at the Bottom of the Internet of Things: A Review,” in *IEEE Sensors Journal,* vol. 19, no. 9, pp. 3179-3190, 2019, doi:10.1109/JSEN.2019.2891911.

[8] M. Mehrabi, D. You, V. Latzko, H. Salah, M. Reisslein and F. H. P. Fitzek, “Device-Enhanced MEC: Multi-Access Edge Computing (MEC) Aided by End Device Computation and Caching: A Survey,” in *IEEE Access*, vol. 7, pp. 166079-166108, 2019, doi:10.1109/ACCESS.2019.2953172.

[9] H. Xing, L. Liu, J. Xu, and A. Nallanathan, “Joint Task Assignment and Wireless Resource Allocation for Cooperative Mobile-edge Computing,” in *Proc. IEEE International Conference on Communications (ICC)*, USA, 2018. pp. 1–6, doi:10.1109/ICC.2018.8422777.

[10] H. Xing, L. Liu, J. Xu, and A. Nallanathan, “Joint Task Assignment and Resource Allocation for D2D-enabled Mobile-edge Computing,” in *IEEE Transactions on Communications,* vol. 67, no. 6, pp. 4193–4207, 2019, doi:10.1109/TCOMM.2019.2903088.

[11] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, “Exploiting Massive D2D Collaboration for Energy-efficient Mobile Edge Computing,” in *IEEE Wireless Communications*, vol. 24, no. 4, pp. 64–71, 2017, doi:10.1109/MWC.2017.1600321.

[12] R. Chai, J. Lin, M. Chen, and Q. Chen, “Task Execution Cost Minimization-based Joint Computation Offloading and Resource Allocation for Cellular D2D Systems,” in *Proc. IEEE International Symposium Personal, Indoor and Mobile Radio Communications*, Italy, 2018, pp. 1–5, doi:10.1109/PIMRC.2018.8580887.

[13] U. Yaqub and S. Sorour, "Multi-Objective Resource Optimization for Hierarchical Mobile Edge Computing," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, United Arab Emirates, 2018, pp. 1-6, doi:10.1109/GLOCOM.2018.8648109.x

[14] M. Mudassar, Y. Zhai, L. Liao and J. Shen, “A Decentralized Latency-Aware Task Allocation and Group Formation Approach With Fault Tolerance for IoT Applications,” in *IEEE Access*, vol. 8, pp. 49212-49223, 2020, doi:10.1109/ACCESS.2020.2979939.

[15] T. Le Duc, R. G. Leiva, P. Casari, and P. O. Östberg, “Machine Learning Methods for Reliable Resource Provisioning in Edge-cloud Computing: A Survey,” in *ACM Computing Surveys*, vol. 52, no. 5, pp. 1–39, 2019, doi:10.1145/3341145.