I-BOT: INTERFERENCE BASED ORCHESTRATION OF TASKS FOR DYNAMIC UNMANAGED EDGE COMPUTING

A Thesis

Submitted to the Faculty

of

Purdue University

by

Shikhar Suryavansh

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Prof. Saurabh Bagchi, Co-Chair
School of Electrical and Computer Engineering

Prof. Mung Chiang, Co-Chair
School of Electrical and Computer Engineering

Prof. Xiaojun Lin
School of Electrical and Computer Engineering

Prof. Somali Chaterji
Agricultural and Biological Engineering

Approved by:

Dr. Dimitrios Peroulis, Head of the Graduate Program
School of Electrical and Computer Engineering

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Prof. Saurabh Bagchi for providing me with this research opportunity. His guidance and insights substantially enriched my research journey at Purdue University. Without his continuous support and encouragement, the goal of this project would not have been realized. I also cherish the friendly chats we had that helped me gain a unique perspective on the life in academia and the industry.

I would like to extend my sincere thanks to my co-advisor, Prof. Mung Chiang for providing valuable feedback throughout the course of this project. The meetings and conversations with him played an important role in thinking outside the box and coming up with innovative ideas.

I am also grateful to the other members of my advisory committee, Prof. Xiaojun Lin and Prof. Somali Chaterji for providing valuable inputs and help throughout my graduate study. My heartfelt thanks to all the members of ECE graduate office, especially Matt Golden who was always there for any help and clarification during my Master's program. Also, I would like to acknowledge the financial support provided by NSF that funded this project.

I wish to thank my fellow labmates at DCSL: Heng Zhang, Atul Sharma, Ran Xu and Manish Nagraj, for the stimulating discussions and advice which helped me in shaping my ideas better. Also, I would like to thank my friends at Purdue University: Ajinkya Mulay, Chandan Bothra, Poorva Parande, Rujuta Barve and Yash Gugale, for always being there for me and making this journey fun and enjoyable.

Last, but not the least, I would like to thank my parents and my brother for always being my support system and constantly motivating me. This journey would not have been possible without them.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Suryavansh, Shikhar MS, Purdue University, August 2020. I-BOT: Interference Based Orchestration of Tasks for Dynamic Unmanaged Edge Computing. Major Professor: Saurabh Bagchi.

The increasing cost of cloud services and the need for decentralization of servers has led to a rise of interest in edge computing. In recent years, edge computing has become a popular choice for latency-sensitive applications like facial recognition and augmented reality because it is closer to the end users compared to the cloud. However, the presence of multiple edge servers adversely affects the reliability due to difficulty in maintenance of heterogeneous servers. In this thesis, we first evaluate the performance of various server configuration models in edge computing using EdgeCloudSim, a popular simulator for edge computing. The performance is evaluated in terms of service time and percentage of failed tasks for an Augmented Reality application. We evaluated the performance of the following edge computing models, Exclusive: Mobile only, Edge only, Cloud only; and Hybrid: Edge & Cloud hybrid with load-balancing on the Edge, and Mobile & Edge hybrid. We analyzed the impact of variation of different parameters such as WAN bandwidth, cost of cloud resources, heterogeneity of edge servers, etc., on the performance of the edge computing models. We show that due to variation in the above parameters, the exclusive models are not sufficient for computational requirements and there is a need for hybrid edge computing models.

Next, we introduce a novel edge computing model called unmanaged edge computing and propose an orchestration scheme in this scenario. Although infrastructure providers are working toward creating managed edge networks, personal devices such as laptops, desktops, and tablets, which are widely available and are underutilized,

can also be used as potential edge devices. We call such devices *Unmanaged Edge Devices (UEDs)*. Scheduling application tasks on such an unmanaged edge system is not straightforward because of three fundamental reasons—heterogeneity in the computational capacity of the UEDs, uncertainty in the availability of the UEDs (due to the devices leaving the system), and interference among multiple tasks sharing a UED. In this work, we present I-BOT, an interference-based orchestration scheme for latency-sensitive tasks on an Unmanaged Edge Platform (UEP). It minimizes the completion time of applications and is bandwidth efficient. I-BOT brings forth three innovations. First, it profiles and predicts the interference patterns of the tasks to make scheduling decisions. Second, it uses a feedback mechanism to adjust for changes in the computational capacity of the UEDs and a prediction mechanism to handle their sporadic exits, both of which are fundamental characteristics of a UEP. Third, it accounts for input dependence of tasks in its scheduling decision (such as, two tasks requiring the same input data). To demonstrate the effectiveness of I-BOT, we run real-world unit experiments on UEDs to collect data to drive our simulations. We then run end-to-end simulations with applications representing autonomous driving, composed of multiple tasks. We compare to two basic baselines (random and round-robin) and two state-of-the-arts, Lavea [SEC-2017] and Petrel [MSN-2018] for scheduling these applications on varying-sized UEPs. Compared to these baselines, I-BOT significantly reduces the average service time of application tasks. This reduction is more pronounced in dynamic heterogeneous environments, which would be the case in a UEP.

1. INTRODUCTION

A lot of service providers host their applications on the cloud because of the benefits of reliability, scalability and cost-effectiveness. According to a recent survey [1], 90% of the companies are hosted on the cloud. However, with an increase in the number of latency-sensitive applications like artificial intelligence, cloud gaming, and augmented reality, there has been a rising interest in edge computing [2]. Cloud being far from the end users is unable to support the stringent latency requirements of such applications. In the case of edge computing, computational resources are placed closer to the end users, thereby reducing latency. Although hosting applications on the edge is attractive, edge computing adds several challenges for service providers that are distinct from cloud computing. The computational resources on the edge are heterogeneous and may not be as powerful or dependable as the cloud computing servers. Also, the available edge resources are at varying geographical distances from the users and the closest resource may not always provide the optimal service time for the applications. Therefore, when multiple users simultaneously send requests pertaining to different applications in an edge computing scenario, selecting the "best" edge device to serve the requests is non-trivial.

In our work, we first investigate the performance and scalability of various existing edge computing models. In particular, we consider five computing models: three exclusive models: Mobile only, Edge only, Cloud only and two hybrid models: Edge & Cloud hybrid with load-balancing on the Edge, and Mobile & Edge hybrid. We use EdgeCloudSim [3], an open source simulator for edge computing with possibilities to conduct experiments based on both computational and networking resources. We focus on an Augmented Reality application in EdgeCloudSim and consider the impact of the variation of the following parameters:

- WAN and WLAN bandwidth: These help in controlling the rate of data transfer from the client to the servers, either edge or cloud.
- Number of edge servers: It is used to simulate edge failures. As the edge servers are not as robust as the cloud, a lot of failures can occur leading to a reduction in the number of available edge servers.
- Number of cloud hosts: We have used the number of cloud hosts to simulate varying cost of cloud resources under the assumption that if the cloud becomes expensive, the number of affordable cloud hosts would come down.
- Capacity of edge servers: To simulate the heterogeneity in edge servers, we have used edge servers with varying capacity in terms of MIPS (Million Instructions Per Second) rating.
- Number of mobile clients: We evaluate the performance of the models as the number of mobile clients increases thereby putting more load on the network.

We found that the exclusive cloud model is the optimal choice under ideal conditions. However, a high variation in the parameters in the real world could result in constrained conditions such as low WAN bandwidth, high edge server failures, etc. In such scenarios, the exclusive models become unsuitable and we see that the hybrid models perform better.

We then introduce the concept of **"unmanaged edge"**. Terms such as "cloudlets" [4, 5], "micro data centers" [6,7], and "fog" [8,9] have been used in the literature to refer to small, edge-located data centers. These cloudlets are managed by infrastructure providers such as Amazon [10], Cisco [11], and Google [12]. However, with an increasing interest in executing latency-critical applications on the edge, and the personal devices such as laptops/desktops/tablets becoming more powerful than ever, there is a scope of utilizing these devices as potential edge "servers". We call such devices **"Unmanaged Edge Devices" (*UEDs*)**. This can be thought of as moving a step closer to the end users in the user-edge-cloud continuum (shown in Figure 1.1). A possible scenario where unmanaged edge can be useful is in real-time road traffic ana-

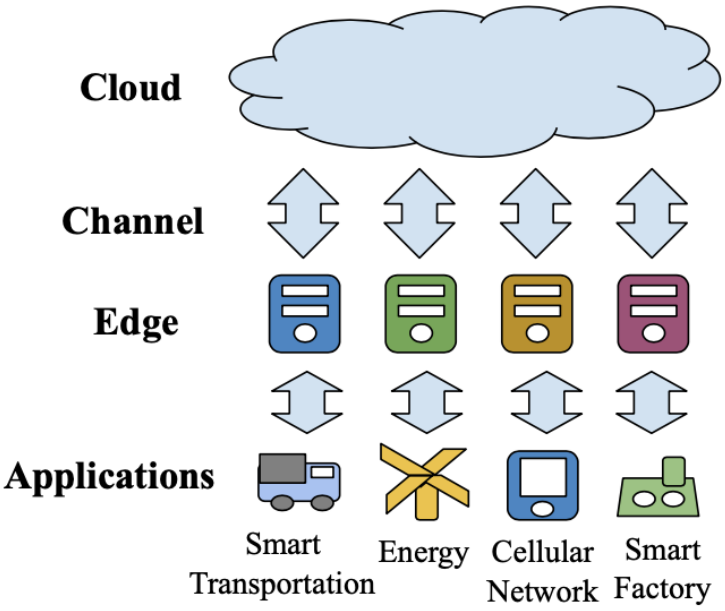


Fig. 1.1.: User-Edge-Cloud continuum

lytics using the video feed from traffic signal cameras. This involves significant video processing in real time to detect events like road accidents, traffic congestion, over-speeding, etc. Since this application is highly latency sensitive, using cloud for the processing would not suffice. In this scenario, the unmanaged edge devices available in the vicinity can be utilized.

Our Solution: I-BOT

In this thesis, we present **I-BOT**, **I**nterference-**B**ased **O**rchestration of **T**asks, for unmanaged edge computing. I-BOT optimizes the service time and the bandwidth utilization of complex applications with multiple tasks that are to be offloaded to the UEDs. We focus on task orchestration in an *unmanaged* edge because of the following reasons. First, managed edge devices are not yet widespread and obviously such infrastructure deployment requires significant cost and efforts to make them ubiquitous. Second, almost everyone today has powerful computing devices which are rarely utilized to their capacity. We propose using these existing underutilized resources instead of investing heavily in the infrastructure for managed edge. We

verified the feasibility of unmanaged edge through a user survey (Figure 6.1) in which 86.4% of the participants indicated their willingness to participate in unmanaged edge computing (under one of four proposed incentive schemes).

There has been a significant amount of work [13–17] on scheduling tasks in a managed edge computing platform. However, since the unmanaged edge devices are not supervised by a particular entity, scheduling tasks to minimize latency in this scenario poses some unique challenges. These include substantial heterogeneity in computational capacity of the UEDs and task interference patterns among co-located tasks on one device, as well as runtime variations in the usable capacity of the edge devices. Also, the UEDs may only be available sporadically and have unpredictable churn. The existing scheduling schemes do not holistically consider all of these unique challenges and hence are not sufficient for task orchestration in an unmanaged setting. Also, many existing works [18–21] utilize the monitoring information provided by the edge devices such as CPU usage, frequency, memory usage, etc. to make decisions regarding which edge device a particular task should be offloaded to. In the case of unmanaged edge, this information may not be readily available due to privacy concerns by the owner of the device, the performance perturbation to collect such monitoring data, and the network cost of conveying that data. Even if it is available, with the amount of added dynamism and heterogeneity introduced by unmanaged edge, the information quickly becomes stale and making decision based on such information may not work. For example, a more powerful laptop may execute a task faster than a less powerful tablet, even if the current CPU usage is much higher for the laptop. A geographically closer computationally less powerful tablet may execute a task faster than a geographically farther more powerful laptop. Additionally, if we factor in the interference caused by co-located applications on a particular unmanaged edge device, decision making based on monitoring individual edge devices may be inaccurate or unscalable. I-BOT overcomes these challenges and minimizes the service time and bandwidth overhead of tasks in the unmanaged edge scenario.

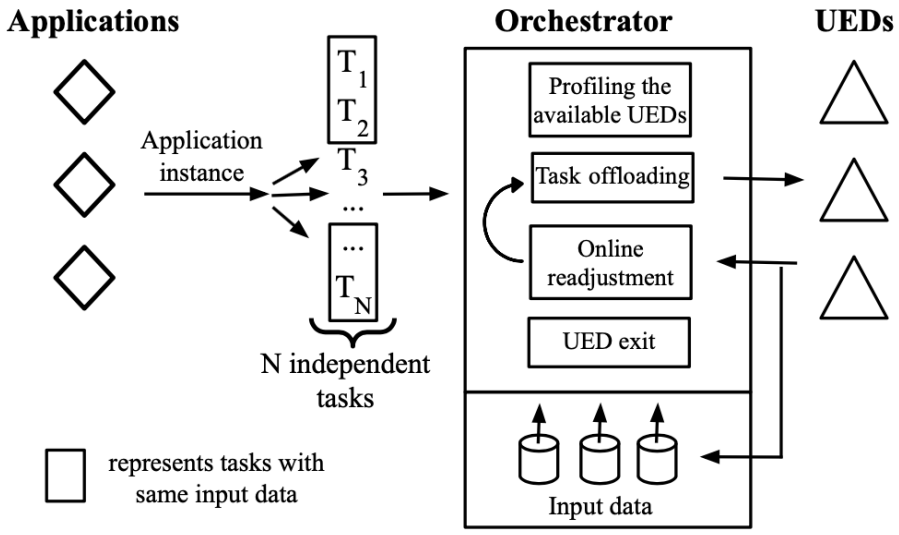


Fig. 1.2.: System Overview

Figure 1.2 presents an overview of the main components of I-BOT. Each application instance from an end user consists of N tasks, some of which are dependent in that they may require the same input data for execution. Application instances are sent to our orchestrator (I-BOT) which schedules the tasks to the available heterogeneous UEDs. The orchestration scheme includes interference profiling of the available UEDs, selecting the optimal UEDs for the execution of tasks based on this profiling information and input parameters (such as the number of tasks running on the UEDs computed based on the number of tasks sent and responses received by the orchestrator), adjusting for the online heterogeneity based on the feedback and an efficient mechanism for UED exit[1]. The UEDs are selected to minimize the service time of the tasks and reduce the bandwidth overhead. Bandwidth overhead can occur if tasks that require the same input data are sent to different UEDs, especially if the input data is huge. I-BOT does not require any monitoring information from the UEDs.

In our evaluation, we compare I-BOT to two intuitive baselines (random and round-robin assignment of tasks) and two state-of-the-art solutions, LAVEA [14] and Petrel [13]. Compared to the existing schemes, I-BOT significantly reduces the aver-

[1]For simplicity of exposition, we describe the orchestrator as if it is centralized. In practice, standard fault-tolerance replication techniques can be used to make it distributed and fault-tolerant [22, 23].

age service time of application instances by at least 61% (Figure 6.2). The reduction in average service time is more significant in the presence of online heterogeneities such as variation in the computational capability of UEDs (Figure 6.3) or sporadic availability of UEDs (Figure 6.4). At the same time, the bandwidth overhead for I-BOT is at least 56% lower than that of the other schemes (Figure 6.5).

Contributions: Our contributions in this paper can be summarized as follows.

1. We present I-BOT, an interference-based dynamic task orchestration scheme to execute user applications consisting of multiple tasks in a heterogeneous unmanaged Edge computing environment. I-BOT optimizes for latency and bandwidth overhead in a configurable manner.

2. Our proposed orchestration scheme takes into consideration the heterogeneity in interference patterns across multiple $UEDs$, the sporadic availability of $UEDs$, and the runtime variations in their computational capacity due to co-located applications. It does not require any monitoring information from the UEDs.

3. We perform extensive simulations and real-world evaluations to demonstrate the effectiveness of I-BOT over four baseline solutions.

The rest of the thesis is organized as follows: Section 2 provides the performance evaluation of various existing edge computing models. Section 3 presents the motivation for unmanaged edge computing and the main challenges involved in task orchestration in this scenario. Section 4 provides a high-level overview of the system components. Section 5 presents the design of the proposed orchestration scheme and Section 6 the evaluation results. Section 7 elaborates on extensions to our work. Finally, Section 8 discusses related work and Section 9 concludes the thesis.

2. PERFORMANCE EVALUATION OF EDGE COMPUTING MODELS

2.1 Methodology

In this section, we describe the different existing edge computing models and provide details about EdgeCloudSim, the simulator used for our simulations. We also describe the performance metrics used for comparison of the models.

2.1.1 Edge Computing Models

The current Mobile-Cloud computing model is a two-tier architecture. Edge computing augments this model by providing a third tier. Figure 2.1 presents five types of computing models possible with edge computing. As an extension of EdgeCloudSim, we focus on these models: T1 (Mobile only), T2 (Edge only) and T3 (Cloud only) are the exclusive models where the computation takes place only on the Mobile, Edge or Cloud servers respectively. With the increase in computing capacity of mobile devices, a large number of applications can be run on the mobile device itself (T1). However, applications such as Augmented Reality, Computer Vision, etc. implement high computational features such as scene understanding, object recognition and object classification [24]. Hence, there is a need to communicate with a Cloud (or Edge) to satisfy the higher computational demands. The performance of a particular exclusive model may vary a lot with the network conditions and other parameters. Therefore, we consider two hybrid models as well: T4 and T5. In T4, the Edge & Cloud hybrid model, the edge servers can also offload tasks to the Cloud, whenever required. T4 which is built on top of T2 offers more flexibility by load-balancing jobs among multiple edge servers and the cloud server. Finally we consider a Mo-

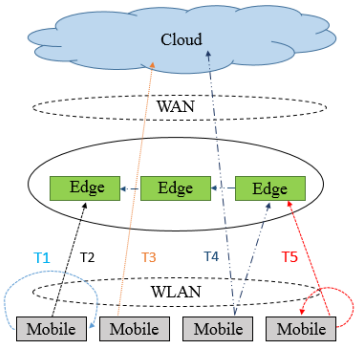


Fig. 2.1.: Models in Edge Computing: T1 (Mobile only), T2 (Edge only), T3 (Cloud only), T4 (Edge & Cloud hybrid) and T5 (Mobile & Edge hybrid)

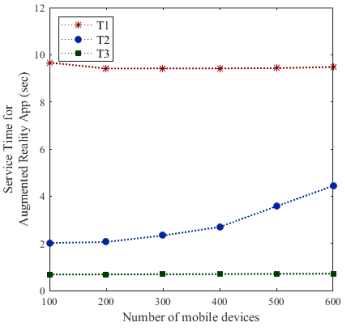


Fig. 2.2.: Comparison of service time of tasks in exclusive Mobile, Edge and Cloud models in ideal conditions

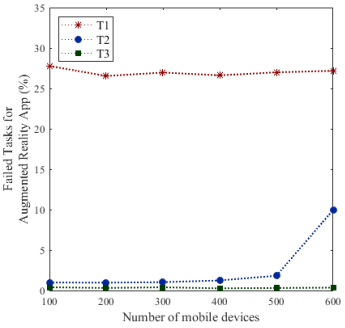


Fig. 2.3.: Comparison of percentage of failed tasks in exclusive Mobile, Edge and Cloud models in ideal conditions

bile & Edge hybrid model T5 consisting of mobile devices and edge servers, shifting the paradigm from Mobile-Cloud model to Mobile-Edge model. It illustrates how a balance between computation at mobile devices and edge servers could yield better service time and less failure of tasks.

2.1.2 EdgeCloudSim

To simulate the edge computing models, control the parameters and evaluate the performance metrics, we need a simulator which can handle the network delays, manage the location of edge and mobile devices, provide a utilization model for Edge Virtual Machines (VMs) and an orchestrator to distribute the incoming tasks.

EdgeCloudSim [3] is a recent simulator which provides these features and has been designed for performance evaluation of edge computing systems. EdgeCloudSim is based on CloudSim [25], which is a mature cloud computing simulation framework.

2.1.3 Performance Metrics

We have used service time as one of the performance metrics for comparison of various models. Service time refers to the total time from the initiation of an application request by the client to the time application request is completed. It includes both the network delay and the execution time.

We shall see that Edge Computing has its benefits but also introduces more failure modes into current computing framework. Since we are interested in how edge servers impact the performance, we will mainly focus on the additional failure modes brought by edge computing and look at the percentage of failed tasks as another metric for the comparison of the models. The tasks can fail due to lack of VM capacity or poor network bandwidth. If the VM utilization is too high, new tasks cannot be accepted leading to failures. Due to limited network capacity, tasks may be dropped if too many clients connect to the same access point leading to failures.

2.2 Comparison of Exclusive Mobile, Edge and Cloud Models in Ideal Conditions

In this section we compare the performance of exclusive Mobile, Edge and Cloud configuration models considering ideal conditions. We focus on the Augmented Reality application and observe the service time and percentage of failed tasks metrics. Based on [3], we have chosen values of the parameters under ideal conditions as: WAN bandwidth = 15 Mbps, WLAN bandwidth = 200 Mbps, No. of edge devices = 14, No. of cloud hosts = 4, Capacity of edge devices = 4,000 MIPS, to understand the default performance of the models.

The cloud hosts run on a single data center and each host contains four cloud VMs. The capacity of each cloud VM is 10,000 MIPS which is significantly higher compared to the capacity of edge servers. To obtain the plots, we varied the number of mobile devices from 100 to 600, in intervals of 100. Figure 2.2 shows the obtained service time plot.

From Figure 2.2, we can see that under ideal conditions, Cloud outperforms Edge and Mobile by a huge margin. Cloud performs the task in a little less than 1 second compared to Mobile which takes almost 10 seconds. Moreover, the time taken remains constant for Mobile and Cloud irrespective of the number of mobile devices. This is because under ideal conditions Cloud has abundant computational resources to handle 600 mobile clients simultaneously. For the exclusive Mobile model, the time remains constant because an increase in the number of mobile devices increases both the count of tasks and the number of mobile servers at the same rate. In the case of the exclusive Edge model, we see an increase in the service time from 2 to 4 seconds as the number of mobile devices increases. This is because the edge devices are limited in number and do not have as high resources as the Cloud. Therefore, an increase in the number of mobile clients results in exhaustion of resources and hence, an increase in the service time for the exclusive Edge model.

A similar trend is observed for the percentage of failed tasks as shown in Figure 2.3. The percentage of failed tasks in exclusive Cloud and Edge models is much lower compared to that in the Mobile model under ideal conditions. It is because the Edge and Cloud have sufficient computational capacity whereas the mobile has limited capacity and will not be able to handle multiple simultaneous task requests.

These results show that for ideal values of the parameters, cloud computation becomes an obvious choice. However, in the real world, there is a persistent variation in the parameters which cannot be ignored. For example, the WAN bandwidth may vary drastically from 1 Mbps to 17 Mbps [26]. Such variations in the parameters can result in a degradation in the performance of the Cloud (or Edge) thereby creating a need for better models. In Section 2.3, we focus on such variations in the parameters which create a need for better hybrid models.

2.3 Need for Hybrid Models

In this section, we explore the performance of the exclusive Mobile, Edge and Cloud models under constrained conditions and explain the need for hybrid models to facilitate good performance in terms of service time and percentage of failed tasks. As evident from Figure 2.2 and Figure 2.3, Cloud performs the best under normal conditions. However, we need to look at the hybrid models (different combinations of Mobile, Edge and Cloud) because of the dynamic changes in the network and the possibilities of failures.

The exclusive Mobile model is independent of the network conditions or the other parameters of interest to us. However, both Edge and Cloud models are susceptible to a change in the parameters. We show that the performance of both Edge and Cloud can degrade and they may become unsuitable under certain conditions.

2.3.1 Unsuitability of Cloud

In this section, we look at the variation of WAN bandwidth and cost of Cloud to understand the degradation in the performance of Cloud.

Due to reduction in WAN bandwidth: A reduction in WAN bandwidth leads to an increase in transmission delay which is critical in applications with high data transfer such as Augmented Reality. Figure 2.4 shows the service time curves for Cloud under different WAN bandwidth (1, 5, 10 and 15 Mbps) conditions, along with the curves for Edge and Mobile. When WAN bandwidth becomes very low (1 Mbps), the service time for Cloud is much higher than that for Edge. A similar degradation was also observed in terms of the percentage of failed tasks.

Due to increase in the cost of the Cloud: The cost of the Cloud is an important factor because with the high prices of cloud computation not everyone can afford extensive cloud resources. We have simulated the cost of Cloud by varying the number of cloud hosts (1, 2, 3 and 4) with the assumption that if Cloud is cheap, one can

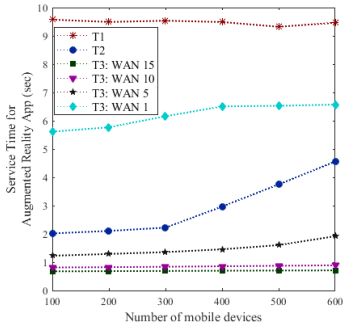


Fig. 2.4.: Impact of reduction in WAN bandwidth on Cloud performance

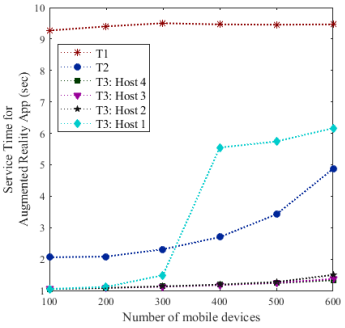


Fig. 2.5.: Impact of increase in cost on Cloud performance

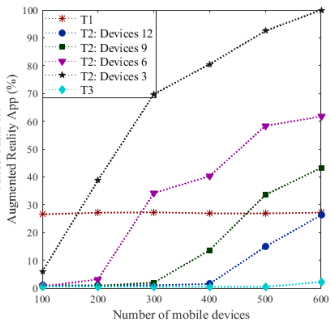


Fig. 2.6.: Impact of failure of edge servers on Edge performance

afford 4 cloud hosts. However, as the cost increases, the number of affordable cloud hosts comes down.

Figure 2.5 shows the service time curves for Cloud as the cost varies along with the curves for Edge and Mobile. As Cloud becomes very expensive (1 cloud host), with a rise in the number of mobile devices, service time for Cloud becomes worse than Edge due to limited computational resources. A similar trend is also observed for the percentage of failed tasks.

2.3.2 Unsuitability of Edge

In this section, we study the unsuitability of Edge due to failure of edge servers and heterogeneity in edge servers.

Due to failure of edge servers: There are Edge devices which are commercially deployed, such as AWS IoT Greengrass [27] which extends AWS to Edge devices so that they can act locally on the data they generate while using the cloud for management. Such deployments, though more reliable, are expensive and not commonly used [28]. In general, the Edge devices are not as well maintained as the centralized cloud servers and are more prone to failures [29]. Hence, the availability of the edge servers can fluctuate drastically. A reduction in the number of available Edge servers

due to failures would result in a computational overload on the functional servers and hence a degradation in the performance. Figure 2.6 shows that an increase in the number of edge failures results in a higher percentage of failed tasks as the number of mobile devices increases. Similarly, the service time for Edge becomes too high with an increase in Edge failures.

Due to heterogeneity in edge servers: A major aspect where edge computing differs from cloud computing is the heterogeneity in edge servers. The computational resources of edge servers may vary from high capacity to low. This makes it difficult to manage the execution of computationally intensive tasks on the Edge. Figure 2.7 shows the plot of the service time for edge servers with varying MIPS rating (from 1000 to 4000). As evident from Figure 2.7, with a reduction in the capacity of the edge servers, the service time of the tasks increases. As the capacity becomes lower than the capacity of Mobile, the service time on Edge can even become worse than the service time on Mobile.

2.4 Performance of Hybrid Models

We showed in Section 2.3 that the performance of both Cloud and Edge depends upon the variation of the parameters. Hence, relying on just the exclusive models is not suitable. We need hybrid models which can perform reliably under a variety of constraints. We perform the evaluation of the following hybrid models: Edge & Cloud hybrid and Mobile & Edge hybrid.

2.4.1 Edge & Cloud Hybrid

We observed that under the constrained conditions in Section 2.3.1 (WAN bandwidth = 1, cloud hosts = 1), the Cloud becomes unsuitable. Figure 2.8 compares the service time of the Edge & Cloud hybrid model with the Cloud model under such conditions.

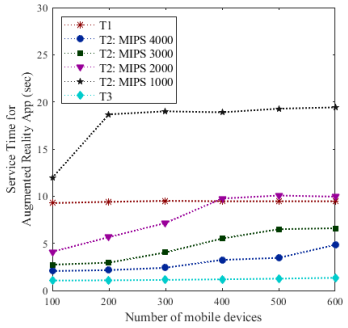


Fig. 2.7.: Impact of varying capacity on Edge performance

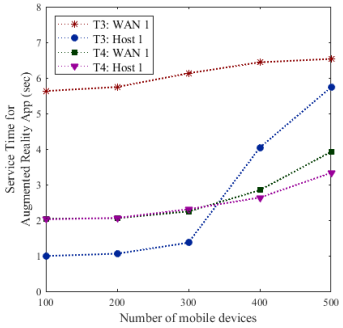


Fig. 2.8.: Comparison of Edge & Cloud hybrid model with Cloud model

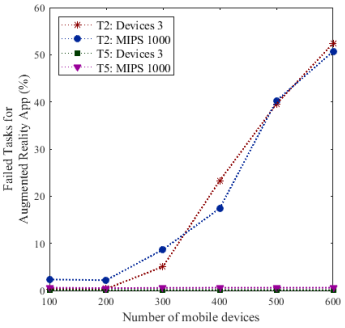


Fig. 2.9.: Comparison of Mobile & Edge hybrid model with Edge model

We see that the Edge & Cloud hybrid model performs better than the Cloud model in terms of the service time. This is because under the constrained conditions, the hybrid model balances the load across the Edge servers and uses Cloud only after the Edge capacity is exhausted which offsets the unsuitability of the Cloud. A similar improvement in performance is also observed for the percentage of failed tasks.

2.4.2 Mobile & Edge Hybrid

The Mobile & Edge hybrid model uses a combination of mobile and edge servers. If the computational resources of the mobile client are not sufficient for the tasks, the tasks are orchestrated to the edge servers. This is useful if the mobile device is not resource rich or is already running many demanding applications. Figure 2.9 compares the percentage of failed tasks in the Mobile & Edge hybrid model with the Edge model under the constrained conditions observed in Section 2.3.2 (Edge devices = 3, Edge MIPS = 1,000).

Under the constrained conditions, the percentage of failed tasks in the Mobile & Edge hybrid model are close to nil compared to the high percentage of failures in the exclusive Edge model as the number of mobile devices increases. It is because the tasks are first executed on the mobile device until the computational limit is hit,

upon which the tasks are directed to the edge servers. Thus, the mobile device and edge servers in the hybrid model receive less traffic compared to the exclusive models leading to lower percentage of failed tasks. We also observed that when the number of mobile clients is high (greater than 500), the hybrid model performs better than the exclusive Edge model in terms of the service time as well.

3. MOTIVATION AND CHALLENGES IN UNMANAGED EDGE COMPUTING

In this section, we consider a motivating example for the unmanaged edge computing scenario and look at the unique challenges introduced by the unmanaged edge.

3.0.1 Motivating Example

Consider a typical application from the domain of autonomous self-driving cars [30]. It has the tasks listed below and we use this application in our evaluation (one of three).

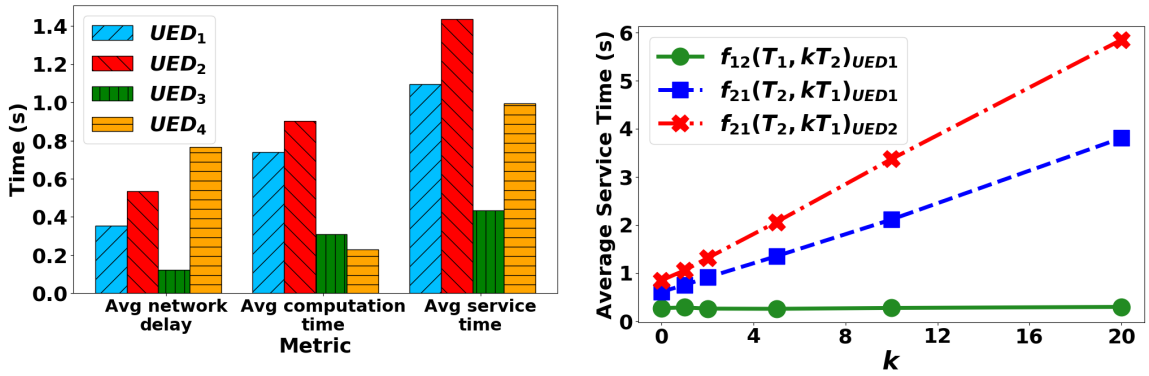
(a) Driver state detection using face camera

(b) Driver body position using driver cabin camera

(c) Driving scene perception using a forward-facing camera

(d) Vehicle state analysis using instrument cluster camera

Task (c) can further consist of multiple tasks like pedestrian detection, obstacle detection, traffic signs analysis, etc. All these tasks would operate on the same input data, *i.e.* the feed from the forward-facing camera. In this work, we focus on how to offload user requests pertaining to the latency-sensitive applications (such as the example above), in a heterogeneous unmanaged edge computing scenario. We aim at minimizing latency while providing a configuration parameter that determines how bandwidth conserving the allocation of tasks to UEDs is.



(a) Computational and geographical heterogeneity

(b) Heterogeneity in interference pattern

Fig. 3.1.: Challenges in unmanaged edge orchestration

3.0.2 Challenges and Responses

The notion of unmanaged edge introduces a set of unique challenges unseen in traditional edge computing. Following are the main challenges involved in the orchestration of tasks in an unmanaged edge scenario and a brief statement about how we handle each challenge.

Substantial heterogeneity in computational capacity and geographical distance of edge devices: The edge devices, which are personal laptops, tablets, desktops, etc., in our case, consist of heterogeneous hardware and hence, the performance of a task varies significantly on different edge devices. Also, different edge devices are at different geographical distances from the orchestrator. Consequently, the network delay also varies. Figure 3.1a shows the average service time (average network delay + average computation time) of executing an image classification task on four heterogeneous edge devices at varying distances from the orchestrator in a production setting. The four UEDs are Samsung Galaxy Tab S4-2018 (UED_1), Dell Inspiron 15R-2013 (UED_2), Macbook Pro-2018 (UED_3) and iMac-2017 (UED_4). Note the huge disparity between the average network delay (max-min ratio 6:1) due to geographical heterogeneity and the average computation time (max:min ratio 4:1) due to computational heterogeneity among the UEDs.

Heterogeneity in task interference pattern: Different tasks, when running on the same edge device, may interfere with each other affecting their service time. There is a heterogeneity in the interference experienced by different types of tasks on a UED. For instance, Figure 3.1b considers task T_1, an image segmentation task, which is simpler compared to T_2, an image classification task. It shows the difference between the interference of tasks of type T_1 on T_2 $\left(f_{21}(T_2, kT_1)_{UED1}\right)$ and T_2 on T_1 $\left(f_{12}(T_1, kT_2)_{UED1}\right)$ on UED_1. The interference is quantified using $f_{ij}(T_i, kT_j)_{UEDp}$ which gives the execution time of a new task of type T_i on UED_p, given that k tasks of type T_j are already running on the UED. It can be seen from the figure that there is a high interference of T_1 on T_2 but almost negligible interference of T_2 on T_1. Not only do different types of tasks interfere differently on the same device, but also there is variation in interference pattern across multiple devices. Figure 3.1b shows the comparison between the interference of T_1 on T_2 on two different $UEDs$ $\left(f_{21}(T_2, kT_1)_{UED1}\right.$ and $\left.f_{21}(T_2, kT_1)_{UED2}\right)$. The interference of T_1 on T_2 is higher on UED_2 than that on UED_1. Thus, interference depends on the ordered pair of tasks and also the UED. I-BOT performs a novel interference profiling of the UEDs to handle this heterogeneity in interference pattern (Section 5.3).

Online variations in the usable capacity of an edge device: Depending upon the personal applications that the owner is running on a UED, the amount of resources available for edge services will vary. To prevent a slowdown of the UED, we need to reduce the usage of the device if the owner starts running a computationally demanding personal application. I-BOT handles this using online readjustment based on a feedback mechanism (Section 5.6).

Lack of monitoring information from edge devices: Most of the current edge orchestration schemes [18–21] utilize monitoring information, such as CPU usage, frequency, memory consumption, etc., from the edge devices to make offloading decisions. However, we do not use any such information because of the following reasons:

1. As the edge devices in our case are not managed by a single entity, the monitoring information may not be readily available. Also, the owners of the devices

may be privacy sensitive about sharing such information with a third party. Note that they have signed up to contribute some compute resources to the unmanaged edge platform, but that can rarely be interpreted to mean that the device owners want the usage on their devices to be monitored.

2. Monitoring a large number of edge devices with the level of frequency needed to be useful would result in a huge overhead. The devices would have to transmit monitoring information continuously as their usable capacity is susceptible to variations, due to co-located applications starting up and other factors that do not occur at a set frequency.

In I-BOT, the orchestrator learns from external observation and predicts the service time of tasks without using any monitoring information from the edge devices (Section 5.5).

Sporadic availability of unmanaged edge devices: Unlike the traditional servers in a managed edge setting which are always available, the availability of an unmanaged edge device would depend upon the owner of that device. Hence, we cannot rely on the device being available for computation all the time. Depending upon the work pattern of the owner of a device, it may be available intermittently at different times of the day. Based on the history of the availability of UEDs, we predict their future availability and use it in our orchestration scheme (Section 5.4).

4. SYSTEM OVERVIEW

In this section, we present a high level overview of the main components of I-BOT. Figure 4.1 shows the timeline exhibiting the steps involved in adding a new UED to the system, orchestrating tasks to the available UEDs, performing online readjustment and gracefully removing a UED when it wishes to exit the system. As shown in Figure 4.1, when a new UED enters the system, our orchestrator profiles it using our novel interference-based profiling method (Section 5.3) and adds it to the UED profile database which stores the profiling information of all the added UEDs. This method of profiling handles the heterogeneity in the computational capabilities and interference patterns among the UEDs. When an application instance (consisting of N different tasks) from an end user arrives at the orchestrator, the orchestrator first predicts which UEDs would be available throughout the execution of the application instance. It then updates the available UED set to include only those UEDs which have a high probability of not leaving the system. This handles the sporadic availability of the UEDs, an inherent characteristic of unmanaged edge computing systems. An initial schedule for the N tasks is then determined using the UED profile database and the data structure containing the number of tasks of different types already running on the available UEDs. This data structure is updated by the orchestrator whenever it sends a new task to a UED or receives an execution result from a UED. The initial schedule is a many-to-one mapping of the N tasks to the available UEDs, aimed at minimizing the service time of the tasks. Next, I-BOT updates the schedule to reduce the bandwidth overhead at the cost of a slight increase in the service time by trying to schedule the tasks that require the same input data on the same UED. I-BOT includes a bandwidth overhead control parameter that manages this trade-off. The tasks are then sent to the selected UEDs. Upon receiving the execution results, the orchestrator sends them back to the end user. It then updates the UED

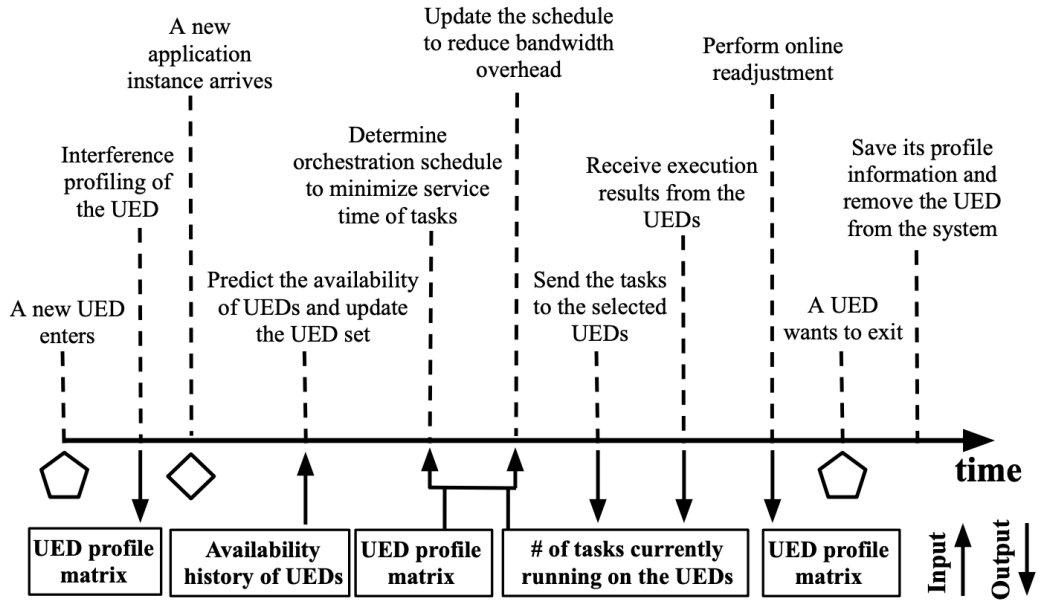


Fig. 4.1.: System Timeline

profile database based on the error between the estimated and actual service time of the tasks on the selected UEDs. The error in the estimation of the service time can occur because of inaccurate profiling of a UED or online heterogeneity such as a variation in the available capacity of a UED. Updating the UED profile based on the feedback error handles such heterogeneities. In the event that a UED wishes to exit the system, its profiling information is saved by I-BOT so that re-profiling is not required whenever the UED re-enters the system.

5. DESIGN

The system consists of our orchestrator running on a *managed* edge device that can offload tasks to multiple $UEDs$ connected to it, as shown in Figure 1.2. The managed edge device is controlled by an infrastructure provider and can be a wireless access point, switch, low to mid range servers installed at the cellular base stations, etc. The end users send application instances to the managed edge device acting as the orchestrator. The orchestrator serves the instances in the order in which they arrive. Our goal is to minimize the total service time of all the tasks in the application instances while reducing the bandwidth overhead. The symbols used in this thesis and their definitions are summarized in Table 5.1.

5.1 Application Structure

Each application instance consists of N tasks, some of which may require the same input data to execute. The structure of a typical application instance is shown in Figure 1.2. It is more bandwidth efficient to send the tasks that require the same input data to the same UED. In our current implementation, we use a linear chain of tasks, though this can be extended to a DAG of tasks with no conceptual novelty (but some engineering effort), as discussed in Section 7.

5.2 Pairwise Incremental Service Time Plots

We define pairwise incremental service time plots $f_{ij}(T_i, kT_j)_p$ to characterize the execution time of a new task of type T_i on UED_p, given that k tasks of type T_j are already running on the UED. This captures the heterogeneity in the interference caused by the tasks. Examples of such plots can be seen in Figures 3.1b and 5.1.

Table 5.1.: Symbols and their definitions.

$i, j \in [1 : N]$; $p \in [1 : N]$

Symbol	**Definition**
$T = \{T_1, T_2, ..., T_N\}$	N different types of tasks for a given application instance
$UED = \{UED_1, UED_2, ..., UED_Q\}$	Q is the total number of UEDs
$f_{ij}(T_i, kT_j)_p = m_{ij} * k + c_{ij} =< m_{ij}, c_{ij} >_p$	Pairwise incremental service time plots on UED_p characterized by slope m_{ij} and y-intercept c_{ij}
$A = [< m_{ij}, c_{ij} >_p]$	Pairwise incremental service time matrix (each row corresponds to a different UED; Figure 5.2)
$Z = [z_{pi}]$	(Task count matrix) Number of tasks of type T_i currently running on UED_p
$ST_{exp}(T_i)_p$	Expected service time of a task of type T_i on UED_p
$ST_{actual}(T_i)_p$	Actual service time of a task of type T_i on UED_p
$R(t)_p$	Probability that UED_p is available continuously between the current time and t time units in the future
Hyper-parameters: (i) δ (ii) β (iii) γ	(i) δ controls the amount of readjustment performed online (ii) β controls the amount of reduction in the bandwidth overhead (iii) γ is minimum threshold for a UED availability for it to be used

We observed that these plots are always straight lines but with varying slopes and y-intercepts due to the task interference and heterogeneity in interference patterns, as elaborated in Section 3.0.2. On a given UED, for a new task T_i, we can plot N pairwise incremental service time plots, one for interference with every other type of task (including T_i). Hence, N^2 such plots exist for every UED and we need to store only N^2 pairs of m and c values to characterize all the plots for that UED. We compute the expected service time of any new incoming task T_i on UED_p, which has $\alpha_1, \alpha_2, \cdots, \alpha_N$ tasks of each type already running using the following equation:

$$f_{i,(1,2,\cdots,N)}\big(T_i,(\alpha_1 T_1,\cdots,\alpha_i T_i,\cdots,\alpha_N T_N)\big) = f_{i1}(T_i,\alpha_1 T_1)+ \\ \cdots + f_{ii}(T_i,\alpha_i T_i)+\cdots+f_{iN}(T_i,\alpha_N T_N). \quad (5.1)$$

This assumes that the interference patterns are independent and additive. We verify this experimentally as can be seen in Figure 5.1. The figure shows that the curve obtained by adding $f_{21}(T_2, jT_1)$ and $f_{22}(T_2, kT_2)$ is very similar to $f_{2,(1,2)}(T_2,(jT_1,kT_2))$.

We define a pairwise incremental service time matrix A, each row of which contains the N^2 pairs of m and c values for a particular UED. See Figure 5.2 for the structure of matrix A. The element $< m_{ij}, c_{ij} >_p$ means that if we want to schedule a new task of type T_i while k instances of task T_j are running on a UED_p, the service time of this task T_i will be estimated as $m_{ij} * k + c_{ij}$. We also define a task count matrix Z, each row of which contains the number of tasks of all the different types currently running on a particular UED. Since the orchestrator sends the tasks and receives the execution results from the UEDs, it keeps updating the matrix Z, whenever needed.

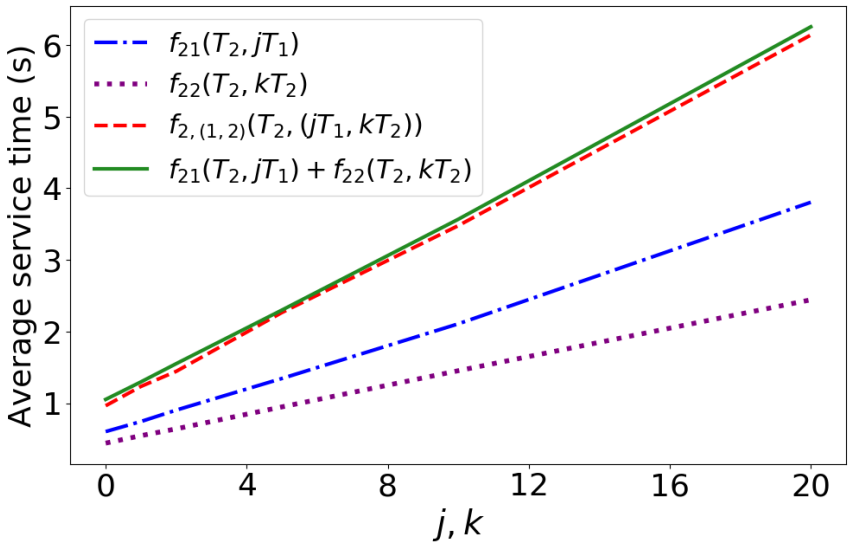


Fig. 5.1.: Experimental validation for computing the expected service time of a new incoming task using Eq. (5.1); j and k are the number of tasks of T_1 and T_2 already running on the UED respectively

$$A_{Q,N^2} = \begin{pmatrix} < m_{11}, c_{11} >_1 & \cdots & < m_{ij}, c_{ij} >_1 & \cdots & < m_{NN}, c_{NN} >_1 \\ < m_{11}, c_{11} >_2 & \cdots & < m_{ij}, c_{ij} >_2 & \cdots & < m_{NN}, c_{NN} >_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ < m_{11}, c_{11} >_p & \cdots & < m_{ij}, c_{ij} >_p & \cdots & < m_{NN}, c_{NN} >_p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ < m_{11}, c_{11} >_Q & \cdots & < m_{ij}, c_{ij} >_Q & \cdots & < m_{NN}, c_{NN} >_Q \end{pmatrix}$$

Fig. 5.2.: Pairwise incremental service time matrix A; Q is the total number of UEDs and N is the total number of different types of tasks in each application instance

Note that, in practice, the application instances arriving at the orchestrator will not be of the same application type. The application instances can be of different types, each consisting of a different set of tasks. At the orchestrator, there will be a separate matrix A for each application type. However, for ease of exposition, we will present our algorithms as if all application instances that arrive belong to a single type of application consisting of N tasks.

5.3 Interference Profiling: Adding a New Unmanaged Edge Device

Adding a new UED to the system requires obtaining all the N^2 pairs of m and c values for the UED and adding them as a new row to matrix A (Figure 5.2). One way to obtain the N^2 pairs is to recreate all the required pairwise interference patterns by actually running tasks on the UED. Since each pairwise interference pattern is a straight line, the m and c values for that pattern can be obtained by extracting any two points on the plot. However, this method of profiling a new UED is not desirable for large N since it would require a lot of time and resources to obtain all N^2 pairs. For some UEDs, the amount of time needed to profile may be in the order of several minutes. Also, since the availability of UEDs in the unmanaged setting is sporadic, spending a lot of time in profiling a UED would be inefficient if the UED is not available for long.

To quickly profile a new UED, we use a technique similar to [31], which relies on Singular Value Decomposition (SVD) and PQ reconstruction. This technique is based

on the algorithm Netflix uses to provide movie recommendations to new users who have only rated a handful of movies. The idea is to find similarities between the new user and the existing users who have rated a lot of movies. We profile the first few UEDs by actually obtaining all the N^2 pairs. Thereafter, for every new UED, we get as many pairs as possible within a fixed time bound (1 minute in our experiments and configurable) and estimate the missing pairs using SVD and PQ-reconstruction. The time complexity of SVD and PQ-reconstruction is linear in N and, in practice, only takes a few milliseconds even for a large N (~ 30). Hence, this scheme is much quicker than obtaining all the N^2 pairs. The inaccuracies in the estimation are handled by online readjustment (Section 5.6).

5.4 UED Availability Prediction

One of the challenges in unmanaged edge computing is the sporadic availability of the UEDs (Section 3.0.2). UEDs may enter or exit the system without prior notice. If a task is scheduled on a UED which is unavailable, or which exits the system before task completion, it would be required to reschedule the task thereby increasing the task completion time. I-BOT predicts the availability of the UEDs and schedules tasks on a UED only if there is a high probability of it being available throughout the task completion. We utilize a semi-Markov Process (SMP) model, similar to [32], to predict the reliability R of a UED. This is the probability of the UED being available throughout a future time window. In an SMP model, the next transition not only depends on the current state (as would happen for a pure Markov model) but also on how long the system has stayed at this state. We observed that the availability pattern of a UED is comparable in the most recent days. Hence, using the availability history of a UED on previous days, we calculate the parameters of the SMP to evaluate $R(t)$, the probability that the UED is available continuously between the current time and t time units in the future. Tasks are scheduled on a

UED only if the probability of it being available throughout the time that it takes to complete the most demanding task in the application is greater than a threshold γ.

5.5 Orchestration Scheme

The orchestration algorithm, the largest part of I-BOT, is shown in Algorithm 1. The algorithm consists of four segments: UED availability prediction, minimum service time scheduling, reduction in the bandwidth overhead, and online readjustment. When a new application instance arrives, we first predict the probability of each UED being available throughout the execution of the application instance. The UEDs for which this probability is lower than a threshold γ are dropped out of the scheduling for the current application instance. The orchestrator maintains a count (in matrix Z) of the number of tasks of different types currently running on the available UEDs. The orchestrator uses this count and the pairwise incremental service time matrix A to predict the service time of the tasks on every available UED and create an initial mapping between the tasks and the UEDs. This mapping assigns each task to a UED on which the expected service time for the task is minimum under the current state of other tasks running on each UED. Predicting the service time of a task involves extracting the corresponding entries from the A matrix and using Eq. 5.1.

Next, the orchestrator tries to reduce the bandwidth overhead by making modifications to the initial schedule. For every group of tasks that require the same input data but are scheduled on different UEDs, the orchestrator tries to schedule them on the same UED to reduce the bandwidth overhead. A change in the assigned UED for a task is made only if the relative increase in its service time due to the change is less than a threshold β, which is the bandwidth overhead control parameter. It decides the trade-off between the bandwidth overhead and the average service time. If β is higher, I-BOT becomes more bandwidth conserving at the expense of higher service time. Finally, the tasks are sent and executed by the assigned UEDs. Upon receiving the execution result, the orchestrator computes the actual service time for

Algorithm 1: *Main_Orchestrator*

Input: A new application instance T
Initialization: UED, A and Z
Let t_{max} be the maximum time to execute the most computationally intensive task on the devices in UED
`// UED availability prediction`
for $UED_p \in UED$ **do**
  Compute $R_p(t_{max})$ using semi-Markov Process (SMP)
  if $R_p(t_{max}) \leq \gamma$ **then**
    Remove UED_p from UED
  end
end
`// Minimum service time scheduling`
for $T_i \in T$ **do**
  for $UED_p \in UED$ **do**
    $ST_{exp}(T_i)_p = GetExpectedServiceTime(i, p)$;
  end
  $ST_{exp}^{min}[i] = \min_p \big(ST_{exp}(T_i)_p\big)$;
  $UED_{sel}[i] = \operatorname{argmin}_p \big(ST_{exp}(T_i)_p\big)$;
end
`// Reduction in bandwidth overhead`
Let $K = [k_1, k_2, ...k_R]$ be a group of tasks which require the same input data
for *every* K **do**
  $ued_1 = UED_{sel}[k_1]$;
  for j = 2, ..., R **do**
    $ued_j = UED_{sel}[k_j]$;
    if $ued_j \neq ued_1$ **then**
      $ST_{min} = ST_{exp}^{min}[k_j]$;
      $ST_1 = GetExpectedServiceTime(k_j, ued_1)$;
      if $\frac{ST_1 - ST_{min}}{ST_{min}} \leq \beta$ **then**
        $UED_{sel}[k_j] = ued_1$;
      end
    end
  end
end
`// Online readjustment`
for $T_i \in T$ **do**
  $p = UED_{sel}[i]$;
  Schedule task T_i on UED_p and compute the actual service time $ST_{actual}(T_i)_p$
  $ST_{exp}(T_i)_p = ST_{exp}^{min}[i]$;
  if $\frac{\left|ST_{exp}(T_i)_p - ST_{actual}(T_i)_p\right|}{ST_{actual}(T_i)_p} > \delta$ **then**
    $PerformGradientDescent\big(i, p, ST_{exp}(T_i)_p, ST_{actual}(T_i)_p\big)$;
  end
end

each task. If the difference between estimated and actual service times for a task is greater than an error threshold (δ), then the orchestrator updates A as described (Section 5.6). For Q total number of UEDs and N tasks in each application instance, the time complexity of our orchestration scheme is $\mathcal{O}(NQ)$. Hence, our scheme can easily scale up without significant overheads.

5.6 Online Readjustment

Online readjustment of the p^{th} row of matrix A is needed when there is a large difference (greater than δ) between the expected and the actual service time of a task T_i on UED_p. This difference arises if there is an inaccuracy in the N incremental service time pairs $< m, c >$ corresponding to T_i in the p^{th} row of A. Following are the main reasons for the inaccuracy:

Imperfect information: As described in Section 5.3, most of the $< m, c >$ pairs in the row added for a UED are computed using SVD and PQ reconstruction and may not be completely accurate.

Online variation: Even if all the $< m, c >$ pairs are correctly profiled initially, the true values may change over time if the owner of the UED starts using a larger portion of the device's compute capability for his/her personal applications. This will result in a change in the pairwise incremental service time plots, thereby changing the $< m, c >$ values.

Algorithm 2: $PerformGradientDescent(i, p, ST_{exp}, ST_{actual})$

Input: $i, p, ST_{exp}, ST_{actual}$
$M = [< m_{ij} >_p]$;
$C = [< m_{ij} >_p]$; $j \in 1, 2, ..., N$
// M and C extracted from p^{th} row of A
$X = TaskCountUED_p = Z[p, :] = [Z_{pj}]; j \in 1, 2, ..., N$
// p^{th} row of Z
$M^{new}, C^{new} = GradientDescent(M, C, X, ST_{actual}, ST_{exp})$;
Update A with M^{new} and C^{new};

Therefore, we need to make online adjustments to the matrix A. For this, we use gradient descent as described in Algorithm 2. For a task T_i scheduled on UED_p, if the difference between the expected and the actual service time exceeds δ, gradient descent is performed to minimize the error between the expected and actual service time and obtain the new values of $< m, c >$ for task T_i on UED_p.

5.7 Unmanaged Edge Device Exit

A UED may leave the system if there is a sudden unexpected crash or if the owner of the UED exits the system. Not much can be done in the case of an unexpected crash. However, in the other case, we perform an additional step for a graceful exit which can save us from re-profiling the UED if it re-joins the system in the future. When the owner of the UED_p wants to exit the system, the information corresponding to the UED stored in the p^{th} row of the A matrix is saved by the system. The row can then be removed from A in the orchestrator. Later, if the UED rejoins the system, its profiling information can be loaded to the orchestrator during the entry phase which significantly reduces the time needed to profile the UED on the system.

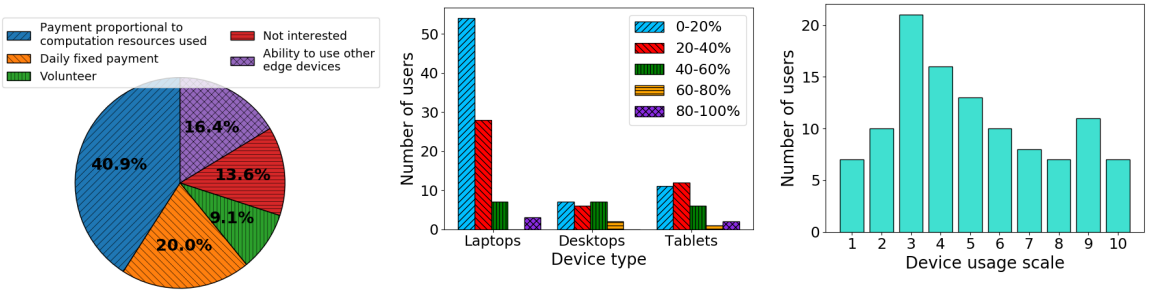
6. EVALUATION

In this section, we first present the major findings from a survey conducted to understand the feasibility of unmanaged edge computing. Then, we provide details about the real-world experiments and how we used them to drive our simulations. We compared the service time obtained by I-BOT with two baseline schemes and two state-of-the-arts for a latency sensitive application. The aim was to evaluate whether our scheme reduces the average service time of the application without a considerable increase in the bandwidth and the orchestration overhead. We then performed a set of micro experiments to evaluate the effect of various control parameters. We then show that our solution works for two other applications, of light and medium load compared to the autonomous driving one introduced earlier.

6.1 Feasibility of Unmanaged Edge: A Survey

We surveyed 110 participants — from USA and India engaged in diverse fields such as educators, software professionals, students, engineering professionals, etc. — to understand the feasibility of unmanaged edge computing. 86.4% of the participants indicated their willingness to provide their computing devices (*e.g.* laptops, desktops, tablets, etc.) as UEDs under one of four proposed incentive models. Only 13.6% of the participants were not interested primarily because of privacy and security concerns. The major takeaways from the survey are the following, as shown in Figure 6.1:

Preferred incentive model: As expected, the majority (40.9%) of the participants were willing to contribute their devices for edge computing if they received a payment proportional to the computational resources of their devices used, as shown in Figure 6.1a. Daily fixed payment (20%) and the ability to use other Edge devices for their applications (16.4%) were second and third most popular choices respectively.



(a) Preferred incentive model (b) Percentage of CPU resources willing to share (c) Device usage (1: Low usage, 10: Very high usage)

Fig. 6.1.: User survey results (Number of participants = 110)

Percentage of CPU resources willing to share: Most of the participants were willing to share between $0 - 40\%$ of the CPU resources of their devices, as shown in Figure 6.1b. It is interesting to note that for tablets, more people were willing to share $20 - 40\%$ resources as compared to laptop owners who mostly showed a willingness to share $0 - 20\%$ resources. This result indicates that people do not use the computational resources of tablets as extensively as laptops and are willing to share more resources of their tablets.

Device usage and tolerable slowdown: As shown in Figure 6.1c, we obtained a double Gaussian device usage pattern with peaks at 90% and 30% of usage indicating that most people either use their devices very heavily (video editing, running sophisticated software, etc.) or use them only for minor purposes such as browsing, reading, etc. The average usage across all users was 50.9%, thereby supporting our claim that a lot of devices are not utilized to their capacity. The majority of the people indicated that they could tolerate around 30% slowdown of their devices.

6.2 Real-World Experiment

The purpose of this experiment was to ensure that our simulations (described later) are based on real-world application and device data. We performed experiments with 3 different application types: a light-weight, a medium, and a heavy

Table 6.1.: Average service time of tasks for different application types

Application Type		
Light	**Medium**	**Heavy**
color detection $0.06s$	kernel filtering $0.22s$	driver state detection $0.39s$
image segmentation $0.12s$	contour detection $0.25s$	driver body position $0.45s$
edge detection $0.17s$	feature transformation $0.35s$	vehicle state analysis $0.43s$
		pedestrian detection $0.57s$
		obstacle detection $0.60s$
		traffic sign analysis $0.41s$

application. The tasks in each application type and their average service time on a typical UED (the Macbook Pro one in our testbed) are given in Table 6.1. For instance, the heavy application is the autonomous self-driving car application as described in Section 3.0.1. This application consists of 6 tasks, 3 of which require the same input data. We obtained the incremental service time curves on 15 heterogeneous $UEDs$ (laptops, desktops and tablets) by running actual application instances on the $UEDs$. For example, the incremental service time curves shown in Figures 3.1b and 5.1 were obtained by running real tasks on actual $UEDs$. We used this data to drive the simulations to ensure that our simulation results are representative of reality.

6.3 Simulation Setting

Our simulator, built in Python, considers Poisson arrival of application instances with rate λ. Unless otherwise mentioned, we performed simulations with 500 arrivals of the heavy application instances and set the default values of our hyperparameters for the experiments to $\lambda = 3\ arrivals/s$, $\delta = 0.10$, $\beta = 0.15$, and $\gamma = 0.85$. For every arrival, the orchestrator needs to schedule the 6 tasks among the 15 available $UEDs$. We also provide a comparison for different application types in Section 6.7. We compared our scheme with the following orchestration schemes:

LAVEA: Proposed in [14], LAVEA is a system that offloads computation to edge devices, to minimize the service time for low-latency video analytics tasks. They propose multiple task placement schemes for collaborative edge computation. We compare with their Shortest Queue Length First (SQLF) scheme, which performed the best among their task placement schemes in our evaluations. It tries to balance the total number of tasks running on each edge device.

Petrel: Proposed in [13], Petrel is a distributed task scheduling framework for edge, which employs the strategy of "the power of two choices" [33]. In this scheme, two of the available edge devices are randomly selected and the task is sent to the one with lower expected service time. They compute the expected service time using the processor speed of the available $UEDs$.

Round Robin: In this scheme, the tasks of an application instance are sent to the available $UEDs$ one after the other, i.e., the first task is sent to UED_1, the second to UED_2, and so on.

Random: This is the most basic scheme in which each task of an application instance is sent to a randomly chosen UED.

We evaluated two versions of our scheme:

I-BOT-PI: This stands for I-BOT-Perfect-Information. In this scheme, all the $UEDs$ are correctly profiled by obtaining the N^2 (36) pairs of m and c values for every UED. As mentioned in Section 5.3, this may not be desirable in the real-world because of high initialization time and sporadic availability of $UEDs$. This can be considered as the ideal case with lowest service time against which we compare the other schemes.

I-BOT-I^2: This stands for I-BOT-Imperfect-Information. In this scheme, we use SVD and PQ reconstruction to create the incremental service time matrix A (Section 5.3). This profiling is faster and realistic, but the incremental service time matrix so obtained may have inaccuracies, which are handled using online readjustment.

In our evaluation, we used the following **performance metrics:**

Service Time: For an application instance scheduled by the orchestrator, we define service time as the average completion time of the different tasks in the application instance. We define average service time (service time averaged over all the instances) and running average service time (service time averaged over a moving window of 50 instances). For experiments in which there is a high fluctuation in the service time of application instances, plotting individual service time hinders visualization. We use running average service time for such experiments.

Orchestration Overhead: We define orchestration overhead for an application instance as the total amount of time spent by the orchestration scheme to decide where to schedule the instance. The average over all the application instances is defined as the average orchestration overhead.

Bandwidth Overhead: We define bandwidth overhead for an application instance as the percentage of tasks that require the same input data but are sent to different $UEDs$. The average over all the application instances is defined as the average bandwidth overhead.

6.4 Evaluation of the Orchestration Schemes

In this experiment, we compare the running average service time obtained by different orchestration schemes for 500 application instances arriving at rate $\lambda = 3$ instances/sec. It can be observed from Figure 6.2 that the service time for our schemes is significantly lower than that for the others. Note that, as mentioned earlier, I-BOT-I^2 would require online readjustment. To show the impact of online readjustment, we have used online readjustment only in the right half of Figure 6.2 (from application instance 250 onwards). For the left half, i.e, without online readjustment, the average service time for I-BOT-PI and I-BOT-I^2 are $0.39s$ and $0.72s$ respectively. This is 61.39% and 28.71% lower than the next best scheme LAVEA for which the average service time is $1.01s$. Our performance is better because our schemes take into consideration the different interference patterns of tasks across the $UEDs$, which is not

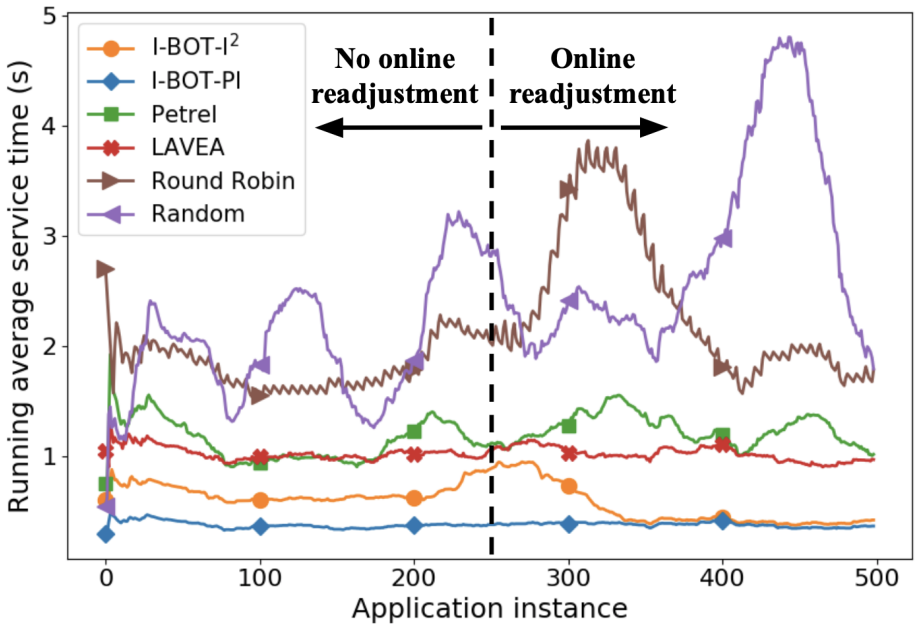


Fig. 6.2.: Comparison of running average service time for different orchestration schemes

considered by the others. For instance, consider two $UEDs$ (UED_1 and UED_2) such that there is a high interference between tasks of type T_1 and T_2 on UED_1 and low interference on UED_2. In this scenario, our schemes will refrain from concurrently scheduling tasks of type T_1 and T_2 on UED_1. LAVEA, on the other hand, would try to equalize the number of tasks of T_1 and T_2 running on the two $UEDs$ thereby resulting in increased interference on UED_1 and a high service time.

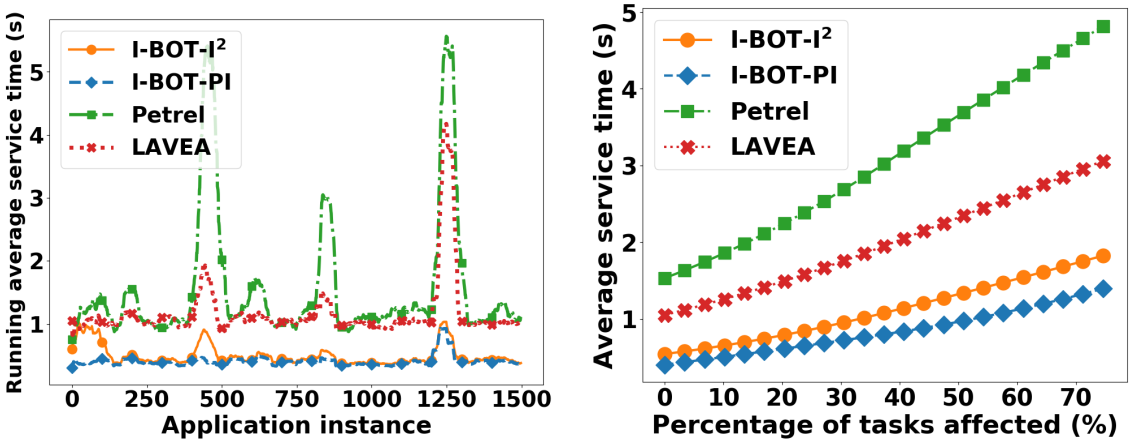
Comparing our schemes with each other, I-BOT-I^2 has inaccuracies in correctly estimating the amount of interference and hence has a higher service time than I-BOT-PI. For I-BOT-I^2, looking at the left and right halves of the figure, we see that it starts with a high average service time but slowly converges to the ideal case. The online readjustment helps not only in alleviating the inaccuracies because of imperfect information but also handles online heterogeneties like variation in the computational capacity and sporadic availability of $UEDs$. These were not considered in Figure 6.2. We present the evaluation with these heterogeneities involved next. The incremental service time matrix constructed by I-BOT-I^2 has an average distance of 0.85 from

the true matrix, distance being computed as the Frobenius norm. For a matrix with values in the range $(0.1, 0.6)$, this can be taken to be a medium level of error.

6.5 Evaluation with Online Heterogeneity

Online heterogeneity happens due to change in the availability of devices online such as if the owner of a particular UED starts/stops running her personal applications resulting in a change in the available computation capacity of the UED or if a UED enters/exits the system.

Impact of co-located applications on the UEDs: Our scheme handles the change in computation capacity of $UEDs$ by continuously updating the incremental service time matrix based on the feedback as explained in Section 5.6. Figure 6.3a shows the comparison of the running average service time for the orchestration schemes with computation capacity of the $UEDs$ varying for 10% of the scheduled tasks. The spikes in the running average service time occur when the available capacity of one or more $UEDs$ suddenly reduces. It is evident from Figure 6.3a that the impact of this variation is the least on our schemes. For other schemes, there is a higher increase



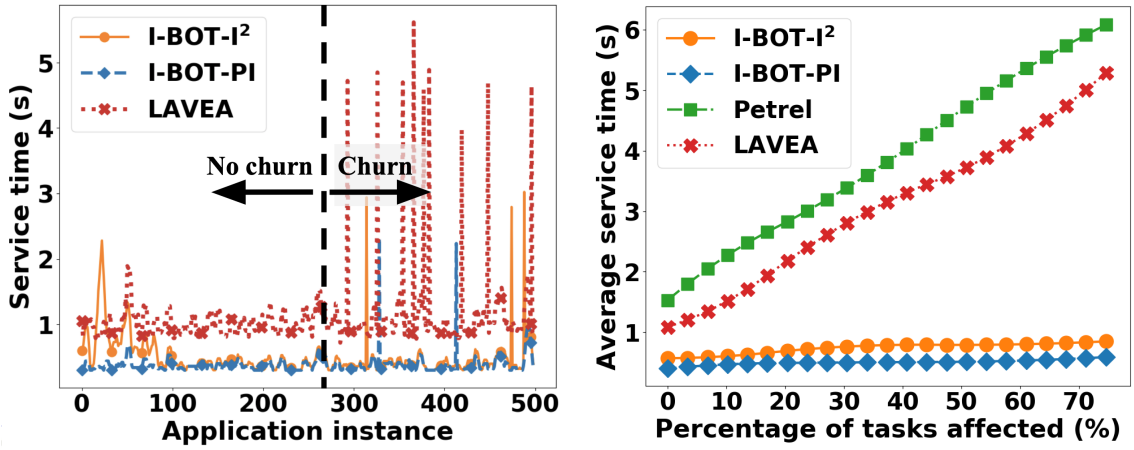
(a) Experiment with variation in computation capacity affecting 10% of the tasks

(b) Impact of changing variation in the computation capacity

Fig. 6.3.: Impact of variation in the computation capacity of the $UEDs$ on the service time

in the service time. The average service time increases by 18.81% and 26.05% for LAVEA and Petrel respectively. On the other hand, the increase is only 7.14% and 9.12% for I-BOT-PI and I-BOT-I^2 respectively. Also, as the amount of variation in the computation capacity increases, a higher percentage of tasks are affected. With this increase in the percentage of affected tasks, the rate of increase in the average service time is much lower for our schemes compared to the others, as shown in Figure 6.3b. We have not shown comparison with round robin and random schemes here because the impact is significantly higher on those schemes.

Impact of churn of UEDs: In Figure 6.4, we show the impact of sporadic availability of $UEDs$ when one or more $UEDs$ abruptly enter/exit the system. Greater the churn of the $UEDs$, higher would be the percentage of tasks affected. Using the availability history of a UED, we predict the probability of the UED being available throughout the task completion. A task is scheduled on a UED only if this probability exceeds the threshold γ. We have used service time for individual instances instead of a running average in this experiment because it better captures the impact of sporadic availability of $UEDs$. In the left half of Figure 6.4a (upto instance 250), all the $UEDs$ are available throughout, whereas in the right half one or more $UEDs$



(a) Experiment with churn affecting 10% of the tasks

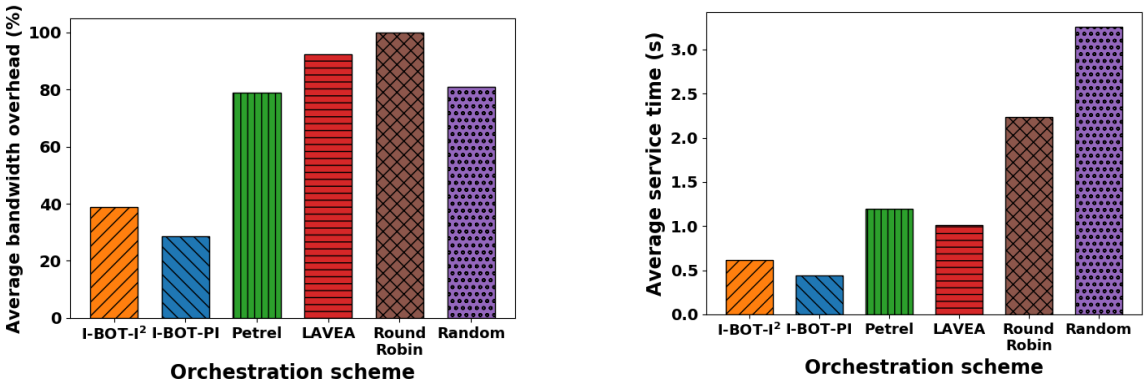
(b) Impact of changing amount of churn of the $UEDs$

Fig. 6.4.: Impact of sporadic availability of the $UEDs$ on the service time

frequently enter/exit the system resulting in 10% of the tasks being affected due to the churn. The spike in service time occurs when one or more tasks of an application instance are scheduled on a UED which is unavailable or which exits the system before the task completion. Since our schemes predict the availability before task scheduling, the spikes are less frequent and shorter compared to the others. The increase in average service time for our schemes with perfect and imperfect information is 5.12% and 7.41% respectively compared to a significantly higher increase of 25.74% and 31.09% for LAVEA and Petrel respectively. In Figure 6.4b, we show the impact of variation in the churn on the average service time. As the churn increases, a higher percentage of tasks are affected. For our schemes, the average service time increases negligibly with an increase in the churn. However, there is a significant increase in the service time for the two other schemes.

6.6 Evaluation of Bandwidth Overhead

The design of our orchestration scheme uses the parameter β that controls the trade off between the average service time and the average bandwidth overhead. In the absence of this design parameter, (i.e., for $\beta = 0$), the average bandwidth overhead for



(a) Average bandwidth overhead

(b) Average service time

Fig. 6.5.: Comparison of average bandwidth overhead and average service time for the orchestration schemes ($\beta = 0.15$ for our schemes)

our scheme with perfect and imperfect information is 82% and 85% respectively (100% means all tasks needing same input data are scheduled on different UEDs). This is comparable to the average bandwidth overhead of the other schemes. However, upon increasing β, there is a considerable reduction in the average bandwidth overhead of our schemes without a significant increase in the average service time. Figure 6.5 shows a comparison of the average bandwidth overhead and average service time of the orchestration schemes for $\beta = 0.15$. The average bandwidth overhead for I-BOT-PI and I-BOT-I^2 reduces to 28.60% and 30.80% respectively, which is much lower than the other schemes. Meanwhile, the average service times for our two schemes do not increase much and are still lower than the others.

6.7 Evaluation with Different Types of Application

Figure 6.6 shows a comparison of the average service time obtained by the orchestration schemes for 500 instances each of the three different application types: light-weight, medium, and heavy (Table 6.1). It is evident from Figure 6.6 that there is an advantage in using our orchestration scheme for all the types of applications. Moreover, this advantage is more pronounced for the heavy application as the tasks involved in such an application have a higher interference with each other and I-BOT schedules the tasks respecting the interference dependencies while the others do not. Most of the latency-sensitive applications that require edge computing belong to this category as they are computationally intensive.

6.8 Evaluation of Orchestration Overhead

Here we compare the average orchestration overhead of the schemes as we vary the total number of $UEDs$. For a given number of $UEDs$, as the stream of application instances arrive, we measure the amount of time spent in making the scheduling decisions for every instance and report the average value over 500 instances. From Figure 6.7, we can observe that the average orchestration overhead for both our schemes

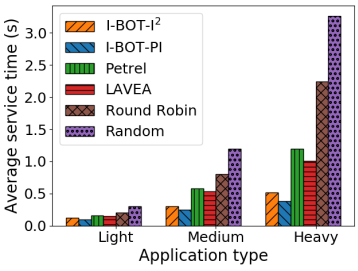


Fig. 6.6.: Evaluation with different types of application

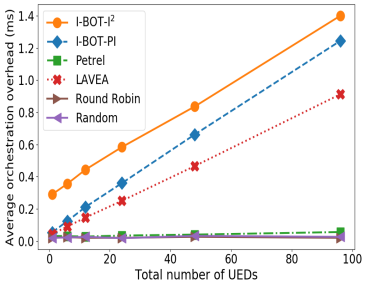


Fig. 6.7.: Evaluation of the orchestration overhead

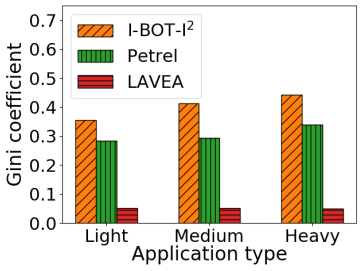
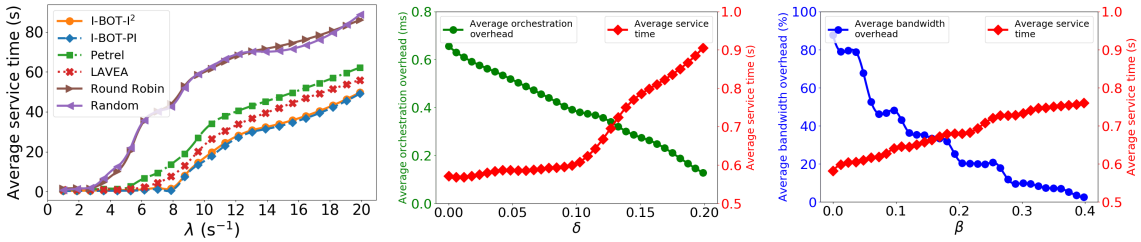


Fig. 6.8.: Evaluation of fairness

is higher than that for the others. However, it is still negligible given the benefits in terms of the reduction in service time. For instance, the average orchestration overhead of I-BOT-I^2 is only $1.4ms$ in the presence of 96 $UEDs$. This accounts for only 0.19% of the average service time of the application instances. Comparing our two variants, the average orchestration overhead is higher for I-BOT-I^2 because of the extra time spent to correct for the inaccuracies due to imperfect information.

6.9 Evaluation of Fairness

In the context of unmanaged edge computing, fairness, defined as balancing the task assignment among multiple $UEDs$, would result in a higher service time because of the substantial heterogeneity among the $UEDs$. We use Gini coefficient to quantify fairness — a value of 0 represents perfect equality whereas 1 represents perfect inequality. Figure 6.8 shows a comparison of the Gini coefficient for I-BOT-I^2, Petrel, and LAVEA for the 3 different application types. As expected, the Gini coefficient is higher for our schemes compared to the others implying a higher inequality in the task distribution among the UEDs. For Petrel, the Gini coefficient is higher than LAVEA, and for round robin, it is equal to 0 as the tasks are perfectly balanced. We argue that this disparity in task allotment is *necessary* for the required improvement in service time because the more powerful UEDs are capable of executing more co-located tasks.



(a) Impact of varying arrival rate (λ) on average service time

(b) Impact of δ on orchestration overhead and average service time of tasks

(c) Impact of β on bandwidth overhead and average service time of tasks

Fig. 6.9.: Micro evaluations

6.10 Micro Evaluations

In this section, we evaluate the impact of varying the parameters of our proposed scheme — application instance arrival rate (λ), readjustment control parameter (δ), and bandwidth overhead control parameter (β). The results are presented in Figure 6.9.

Impact of varying arrival rate (λ) of application instances: For a fixed number of $UEDs$, an increase in the arrival rate (λ) of application instances would ultimately make the system unstable for all the orchestration schemes. However, as shown in Figure 6.9a, this instability occurs in our schemes for a higher value of λ compared to the others, i.e, our schemes can maintain acceptable service time for a higher arrival rate than the other schemes. Also, before the instability sets in, our schemes have lower service time than the others. It is interesting to note that even in the unstable region, the service time for our schemes is lower.

Impact of varying readjustment control parameter (δ): Figure 6.9b shows this evaluation. A low value of δ implies that gradient descent based online readjustment would be invoked more frequently. Thus there is a higher chance of readjusting to the true matrix A, resulting in a lower average service time. However, more readjustment also means a higher orchestration overhead. As δ increases, the average service time curve is flat in the beginning and then increases (for $\delta > 0.1$). This is because while

it is true that the amount of readjustment is lower for $\delta = 0.1$ compared to that for $\delta = 0$, it is still sufficient to correct for the online heterogeneities.

Impact of varying bandwidth overhead control parameter (β): As β increases, the probability of tasks with the same input data being scheduled on the same UED also increases. This results in a reduction in bandwidth overhead as shown in Figure 6.9c. This reduction in the bandwidth overhead comes at the cost of an increase in the service time. However, the increase in the service time is not very significant. For instance, increasing β from 0 to 0.15 reduces bandwidth overhead by 61% whereas the average service time increases only by 10.3%.

6.11 Theoretical Analysis

We present the theoretical analysis of our solution under the following simplifying assumptions. First, we assume that the UEDs are homogeneous and a task of type k has exponentially distributed processing rate μ_k for $k = [1 : N]$, where $\frac{1}{Q}\sum_{k=1}^{N} \lambda/\mu_k < 1$ for irreducible and stable (*i.e.* positive recurrent) Markov chain. Moreover, we assume that tasks of type 1 to N are dispatched to the chosen UED's queue in order. Queue state for UED_q, $q = [1 : Q]$, is then defined by $\phi_n = \{(0)\} \cup \{(t_1, t_2, \ldots, t_n) | n \geq 1\}$, where t_i is the type of the ith task in the (type independent) FIFO order (t_1 is the type of a task being served) and (0) represents the empty system. The processed rate is then determined by the type of a task being served and uniquely determined by queue length i as follows: $\lceil \frac{i}{N} \rceil N - i + 1$. Refer Appendix A.1 for proof of the results shown below.

Lemma 1 *Under our proposed solution, the transition rates $q_{i,j}(\boldsymbol{\pi})$ given distribution $\boldsymbol{\pi}$ for $j \neq i$ is given by*

$$q_{i,j}(\boldsymbol{\pi}) = \begin{cases} \mu_{\lceil \frac{i}{N} \rceil N - i + 1} & \text{if } j = i - 1, \\ \frac{1-(Q-1)\sum_{l=0}^{i-1} \pi_l}{1+(Q-1)\sum_{l=0}^{i} \pi_l} & \text{if } j = i + N,\ i < \tau_{\boldsymbol{\pi}}, \\ 0, & \text{otherwise,} \end{cases}$$

where $\tau_{\boldsymbol{\pi}} = \min\{j : \sum_{l=0}^{j-1} \pi_l \geq \frac{1}{Q-1}\}$ *and* π_l *denotes the stationary distribution of UED queue, i.e., the probability that the queue size is* l *at a UED.*

Intuitively, $\tau_{\boldsymbol{\pi}}$ indicates the queue length so that the probability that a UED with queue size $i(\geq \tau_{\boldsymbol{\pi}})$ receives N tasks is 0. Based on Lemma 1, we can calculate the stationary distribution of the queue length of a single UED numerically by finding $\hat{\pi}$ that satisfies the global balance equation. The expected service time $T_Q(\lambda, \mu_1, \cdots, \mu_N)$ of an application instance that is dispatched to Q UEDs is then

$$\sum_{r=0}^{N-1}\left[\sum_{i=1}^{\infty}\left(\lfloor\frac{i-1}{N}\rfloor\sum_{l=1}^{N}\frac{1}{\mu_l}+\mathbf{1}_{\lceil\frac{i}{N}\rceil N-i+1-r\geq 1}\sum_{m=\lceil\frac{i}{N}\rceil N-i+1-r}^{N-r}\frac{1}{\mu_m}+\right.\right.$$
$$\left.\left.\mathbf{1}_{\lceil\frac{i}{N}\rceil N-i+1-r<1}\sum_{m=N-r}^{\lceil\frac{i}{N}\rceil N-i+1-r+N}\frac{1}{\mu_m}\right)\cdot\left\{\left(\sum_{j=i-1}^{\infty}\pi_j\right)^Q-\left(\sum_{j=i}^{\infty}\pi_j\right)^Q\right\}\right].$$

We compare the expected service times from analysis and simulations, as shown in Figure 6.10, where $N = 2$, $Q = 3$, $\lambda = [1 : 20]$, $\mu_1 = 10$, and $\mu_2 = 30$. It demonstrates that the analytical result serves as a worst case upper bound for the service time as it assumes serial processing, while in reality multiple tasks can be concurrently processed. The worst case will occur in practice if each task is intensive enough to occupy the entire UED. The divergence between analytical and simulation results increases as the load increases, in which case the simulation allows for more parallel processing.

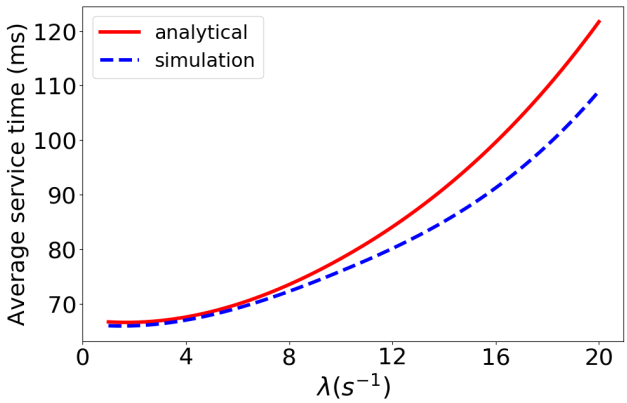


Fig. 6.10.: Comparison of analytical and simulation results

7. DISCUSSION

In this section, we present extensions of I-BOT needed to handle some important cases. *First*, we have only considered one form of dependency among the tasks — same input data requirements for certain tasks. In practice, however, the tasks can also have control dependency, which is often represented by a directed acyclic graph (DAG) of tasks. Our algorithm can conceptually be easily combined with scheduling algorithms that operate over DAGs; there is significant work in the context of conventional computing systems [34,35] and some emerging work in the context of edge computing systems [36]. *Second*, the linearity in the task interference plots may not hold if the number of tasks running on a UED is large enough to cause a discontinuous change. This is a common occurrence with mapping of resource availability to performance metrics (such as, latency) [37], say if the working set of the program spills over from one level of cache into a lower (and higher latency) level of cache. In this case, in I-BOT, a higher-order characterization of the interference plots would be needed (say, quadratic or piece-wise linear) and failure of tasks must also be considered. Next, for simplicity, we fix the order in which the tasks belonging to an application instance are considered for offloading, namely, the same order in which they are enumerated in the application description. This is akin to greedy scheduling with respect to task order and a more optimal scheduling can happen if we use non-local information, such as, through dynamic programming.

8. RELATED WORK

In this section we contrast our work with the other efforts in the field of task scheduling in heterogeneous edge computing systems.

Low latency edge scheduling: Petrel [13] and LAVEA [14] propose orchestration schemes aimed at minimizing the service time in a multi-edge collaborative environment. We have shown that I-BOT outperforms these schemes in terms of the service time and bandwidth overhead in a heterogeneous unmanaged edge computing setting. MSGA [15] jointly studies the task and network flow scheduling and uses a multi-stage greedy algorithm to minimize the completion time of the application. In [17], a gateway-based edge computing service model has been proposed to reduce the latency of data transmission and the network bandwidth. Low latency task scheduling schemes for edge have also been proposed in [38–40]. However, all of these works are in the context of managed edge and do not consider the unique challenges introduced by unmanaged edge, such as the lack of monitoring information, heterogeneity, and unexpected entry-exits. One exception to this is CoGTA [41], which considers scheduling of delay-sensitive social sensing tasks on a heterogeneous unmanaged edge. However, its main focus is on devices that are not trusted and therefore it formulates a game-theoretic technique to perform the task allocation. Its performance in a benign setting like ours is likely to be sub-optimal.

Availability and Interference based edge scheduling: There have been a few efforts that take into account the availability and interference while devising strategies for task scheduling on the edge. An overhead-optimizing task scheduling strategy has been proposed in [18] for ad-hoc based edge computing nodes formed by a group of mobile devices. [19] proposes a score based edge service scheduling algorithm that evaluates network, compute, and reliability capabilities of edge nodes. However, these works rely on sharing monitoring information which can be a huge overhead in highly

dynamic environments. Also, the time and energy consumption models are theoretical and have not been tested on real systems. INDICES [42] proposes a performance-aware scheme for migrating services from cloud to edge while taking into account the interference caused by co-located applications. However, this is geared towards service migration and not task scheduling. Also, it does not consider the impact of online variations in the availability and compute capabilities of edge devices.

Energy efficient edge scheduling: A lot of existing works [43–46] utilize dynamic voltage-frequency scaling (DVFS), which is an attractive method for reducing energy consumption in heterogeneous computing systems. ESTS [47] deals with the problem of scheduling a group of tasks, optimizing both the schedule length and energy consumption. They formulate the problem as a joint linear programming problem and propose a heuristic algorithm to solve it. In [48], a computational offloading framework has been proposed which minimizes the total energy consumption and execution latency by coupling task allocation decisions and frequency scaling. The paper [16] also performs joint optimization of energy and latency through a rigorously formulated and solved mixed integer nonlinear problem (MINLP) for computation offloading and resource allocation. However, the execution models used in these works do not consider the impact of online heterogeneties in the computation capacity or the effect of interference.

Volunteer or opportunistic computing: In a completely different context, under the moniker "volunteer computing", a slew of works designed solutions to utilize under-utilized compute nodes (such as, on a university campus) or mobile devices to run large-scale parallel applications. An example of the former is HTCondor [49] and an example of the latter is Femtocloud [50]. Our design borrows some features from Femtocloud (identifying devices with spare capacity and some stability); however, Femtocloud did not have to deal with the majority of the challenges that we solve here (great heterogeneity from a compute, network, and application standpoint, unknown tasks, runtime variations due to interference).

9. CONCLUSION

In this thesis, we compared the performance of various edge computing models based on parameters such as network bandwidth, computational capacity, etc. We categorized the models as Exclusive and Hybrid depending upon the combination of Mobile, Edge and Cloud servers. We observed that under varying parameters the optimal choice of model can change. As we move from the ideal conditions in which exclusive Cloud model seems to be the obvious choice to scenarios with limited resources, we observe that the hybrid models perform better. We then introduced unmanaged edge computing model and presented a novel Interference Based Orchestration of Tasks (I-BOT) for this model that utilizes personal devices as edge nodes for task execution. We identified three new challenges in orchestrating application tasks in the unmanaged edge scenario, due to which prior edge schedulers fail — heterogeneity in devices, runtime variation in available compute capacity, and sporadic availability of devices. We introduced three design innovations in I-BOT to handle these challenges and thus minimize the service time and bandwidth overhead of latency-sensitive applications. We extensively evaluated our system using real-world experiments and simulations. Results show that compared to existing approaches (two intuitive baselines and two state-of-the-art ones, LAVEA and Petrel), I-BOT significantly reduces average service time and bandwidth overhead of applications by at least 61% and 56% respectively. We also demonstrated that in the presence of online variability, which is an inherent characteristic of unmanaged systems, the reduction in service time and bandwidth overhead due to I-BOT is more prominent.

REFERENCES

REFERENCES

[1] eukhost, "New statistics: Show the advance of cloud computing," https://www.eukhost.com/blog/webhosting/new-statistics-show-the-advance-of-cloud-computing/, 2020, accessed: 2020-06-28.

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[3] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, May 2017, pp. 39–44.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1109/MPRV.2009.82

[5] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: bringing the cloud to the mobile user," in *3rd ACM Workshop on Mobile Cloud Computing and Services, Proceedings*. Ghent University, Department of Information technology, 2012, pp. 29–35.

[6] M. Aazam and E. Huh, "Fog computing micro datacenter based dynamic resource estimation and pricing model for iot," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, 2015, pp. 687–694.

[7] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, p. 68–73, Dec. 2009. [Online]. Available: https://doi-org.ezproxy.lib.purdue.edu/10.1145/1496091.1496103

[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 13–16. [Online]. Available: https://doi-org.ezproxy.lib.purdue.edu/10.1145/2342509.2342513

[9] M. Aazam and E. Huh, "Fog computing and smart gateway based communication for cloud of things," in *2014 International Conference on Future Internet of Things and Cloud*, 2014, pp. 464–470.

[10] "Amazon: Lambda@edge," https://aws.amazon.com/lambda/edge/, 2020, accessed: 2020-06-28.

[11] "Cisco: Establishing the edge," https://www.cisco.com/c/en/us/solutions/service-provider/edge-computing/establishing-the-edge.html, 2020, accessed: 2020-06-28.

[12] "Google: Edge network," https://peering.google.com/#/, 2020, accessed: 2020-06-28.

[13] L. Lin, P. Li, J. Xiong, and M. Lin, "Distributed and application-aware task scheduling in edge-clouds," in *2018 14th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, 2018, pp. 165–170.

[14] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3132211.3134459

[15] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3512–3524, 2019.

[16] J. Zhang, X. Hu, Z. Ning, E. C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2017.

[17] C.-W. Tseng, F.-H. Tseng, Y.-T. Yang, C.-C. Liu, and L.-D. Chou, "Task scheduling for edge computing with agile vnfs on-demand service model toward 5g and beyond," *Wireless Communications and Mobile Computing*, vol. 2018, p. 7802797, Jul 2018. [Online]. Available: https://doi.org/10.1155/2018/7802797

[18] L. Tianze, W. Muqing, Z. Min, and L. Wenxing, "An overhead-optimizing task scheduling strategy for ad-hoc based mobile edge computing," *IEEE Access*, vol. 5, pp. 5609–5622, 2017.

[19] A. Aral, I. Brandic, R. B. Uriarte, R. De Nicola, and V. Scoca, "Addressing application latency requirements through edge scheduling," *Journal of Grid Computing*, vol. 17, no. 4, pp. 677–698, Dec 2019. [Online]. Available: https://doi.org/10.1007/s10723-019-09493-z

[20] A. J. Page and T. J. Naughton, "Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing," in *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 8 pp.–.

[21] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *2017 IEEE International Conference on Edge Computing (EDGE)*, 2017, pp. 47–54.

[22] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Trans. Comput. Syst.*, vol. 1, no. 3, p. 222–238, Aug. 1983. [Online]. Available: https://doi-org.ezproxy.lib.purdue.edu/10.1145/357369.357371

[23] F. B. Schneider and Lidong Zhou, "Implementing trustworthy services using replicated state machines," *IEEE Security & Privacy*, vol. 3, no. 5, pp. 34–43, 2005.

[24] R. Xu, J. Koo, R. Kumar, P. Bai, S. Mitra, S. Misailovic, and S. Bagchi, "Videochef: Efficient approximation for streaming video processing pipelines," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, 2018, pp. 43–56. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/xu-ran

[25] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose, and R. Buyya, "Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services," *CoRR*, vol. abs/0903.2525, 2009.

[26] M. T. Diallo, F. Fieau, and J. Hennequin, "Impacts of video quality of experience on user engagement in a live event," in *2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, July 2014, pp. 1–7.

[27] "Aws iot greengrass," https://aws.amazon.com/greengrass/.

[28] "Aws iot greengrass usage," https://discovery.hgdata.com/product/aws-iot-greengrass.

[29] P. Wood, H. Zhang, M. Siddiqui, and S. Bagchi, "Dependability in edge computing," *CoRR*, vol. abs/1710.11222, 2017. [Online]. Available: http://arxiv.org/abs/1710.11222

[30] L. Fridman, D. E. Brown, M. Glazer, W. Angell, S. Dodd, B. Jenik, J. Terwilliger, A. Patsekin, J. Kindelsberger, L. Ding, S. Seaman, A. Mehler, A. Sipperley, A. Pettinato, B. D. Seppelt, L. Angell, B. Mehler, and B. Reimer, "Mit advanced vehicle technology study: Large-scale naturalistic driving study of driver behavior and interaction with automation," *IEEE Access*, vol. 7, pp. 102 021–102 038, 2019.

[31] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 77–88. [Online]. Available: https://doi-org.ezproxy.lib.purdue.edu/10.1145/2451116.2451125

[32] Xiaojuan Ren, Seyong Lee, R. Eigenmann, and S. Bagchi, "Resource availability prediction in fine-grained cycle sharing systems," in *2006 15th IEEE International Conference on High Performance Distributed Computing*, 2006, pp. 93–104.

[33] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[34] R. Sakellariou and H. Zhao, "A hybrid heuristic for dag scheduling on heterogeneous systems," in *18th International Parallel and Distributed Processing Symposium.* IEEE, 2004, p. 111.

[35] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra, "Dague: A generic distributed dag engine for high performance computing," *Parallel Computing*, vol. 38, no. 1-2, pp. 37–51, 2012.

[36] S. Khare, H. Sun, J. Gascon-Samson, K. Zhang, A. Gokhale, Y. Barve, A. Bhattacharjee, and X. Koutsoukos, "Linearize, predict and place: minimizing the makespan for edge-based stream processing of directed acyclic graphs," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 1–14.

[37] B. Falsafi and T. F. Wenisch, "A primer on hardware prefetching," *Synthesis Lectures on Computer Architecture*, vol. 9, no. 1, pp. 1–67, 2014.

[38] S. Wang, Y. Li, S. Pang, Q. Lu, S. Wang, and J. Zhao, "A task scheduling strategy in edge-cloud collaborative scenario based on deadline," *Scientific Programming*, vol. 2020, p. 3967847, Mar 2020. [Online]. Available: https://doi.org/10.1155/2020/3967847

[39] J. Han and D. Wang, "Edge scheduling algorithms in parallel and distributed systems," in *2006 International Conference on Parallel Processing (ICPP'06)*, 2006, pp. 147–154.

[40] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 365–375.

[41] D. Zhang, Y. Ma, C. Zheng, Y. Zhang, X. S. Hu, and D. Wang, "Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 243–259.

[42] S. Shekhar, A. D. Chhokra, A. Bhattacharjee, G. Aupy, and A. Gokhale, "Indices: Exploiting edge resources for performance-aware cloud-hosted services," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, 2017, pp. 75–80.

[43] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of energy-cognizant scheduling techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1447–1464, 2013.

[44] D. Li and J. Wu, "Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms," in *2012 41st International Conference on Parallel Processing*, 2012, pp. 430–439.

[45] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in dvfs-based energy consumption minimization," *Journal of Parallel and Distributed Computing*, vol. 71, no. 8, pp. 1154 – 1164, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731511000165

[46] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi, "Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster," in *2006 IEEE International Conference on Cluster Computing*, 2006, pp. 1–10.

[47] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 2867–2876, 2014.

[48] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[49] D. H. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne, "A worldwide flock of condors: Load sharing among workstation clusters," *Future Generation Computer Systems*, vol. 12, no. 1, pp. 53–65, 1996.

[50] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *2015 IEEE 8th international conference on cloud computing*. IEEE, 2015, pp. 9–16.

APPENDIX

A. APPENDIX

A.1 Theoretical Analysis

We present the theoretical analysis of our solution under the following simplifying assumptions. First, we assume that the UEDs are homogeneous and a task of type k has exponentially distributed processing rate μ_k for $k = [1 : N]$, where $\frac{1}{Q}\sum_{k=1}^{N} \lambda/\mu_k < 1$. We further assume that tasks of type 1 to N are dispatched to the chosen UED's queue in order. Queue state for UED_q, $q = [1 : Q]$, is then defined by

$$\phi_n = \{(0)\} \cup \{(t_1, t_2, \ldots, t_n) | n \geq 1\},$$

where t_i is the type of the ith task in the (type independent) FIFO order (t_1 is the type of a task being served) and (0) represents the empty system. Under these assumptions, queue length determines the expected service time. Specifically, the expected service time for all tasks in UED is a monotonically increasing function of the UED queue length.

The evolution of the system over ϕ_n is an irreducible Markov chain. Using the Lyapunov theorem, it can be verified that the Markov chain is positive recurrent, and thereby has a unique stationary distribution. Let $\pi(t_1, t_2, \ldots, t_i)$ denote the stationary distribution of UED_q, i.e., the probability that the queue state is $(t_1, t_2, \ldots, t_i)$ at UED_q. Here, the index q is ignored because the stationary distributions are identical across UEDs. We have $\sum_i i\pi(t_1, t_2, \ldots, t_i) = \sum_i i\pi(\phi_i) = \sum_i i\pi_i < c$, where a constant $c > 0$.

Consider the queue evolution of one UED in the system. At steady state, each queue forms an independent Markov chain, as described in the following lemma:

Lemma 2 *Under our proposed solution, the transition rates $q_{i,j}(\boldsymbol{\pi})$ given distribution $\boldsymbol{\pi}$ for $j \neq i$ is given by*

$$q_{i,j}(\boldsymbol{\pi}) = \begin{cases} \mu_{\lceil \frac{i}{N} \rceil N-i+1} & \text{if } j = i-1, \\ \frac{1-(Q-1)\sum_{l=0}^{i-1}\pi_l}{1+(Q-1)\sum_{l=0}^{i}\pi_l} & \text{if } j = i+N,\ i < \tau_{\boldsymbol{\pi}}, \\ 0, & \text{otherwise,} \end{cases}$$

where $\tau_{\boldsymbol{\pi}} = \min\{j : \sum_{l=0}^{j-1}\pi_l \geq \frac{1}{Q-1}\}$ and π_l denotes the stationary distribution of UED queue, i.e., the probability that the queue size is l at a UED.

Proof The transition rates will be determined by our solution used to dispatch tasks to UEDs. We will derive the transition rates for our strategy. First, the down-crossing transition rate from state i to state $i-1$ is

$$\begin{aligned} q_{i,i-1} &= \mu_{t_1} \\ &= \mu_{\lceil \frac{i}{N} \rceil N-i+1} \end{aligned}$$

because the processing time of a task of type t_1 is exponentially distributed with mean μ_{t_1} and the type of a task being served is uniquely determined by queue length i as $\lceil \frac{i}{N} \rceil N - i + 1$ due to our dispatch strategy.

Second, the up-crossing transition rate from state i to state j for $j > i$ is

$$q_{i,j} = \lambda \sum_{\boldsymbol{\eta}} \mathsf{P}(\boldsymbol{\eta}) \cdot \mathsf{P}(j|\boldsymbol{\eta}, i),$$

where $\boldsymbol{\eta}$ is a $(Q-1)$ vector that denotes the queue lengths of the other $Q-1$ UEDs; thus,

$$\mathsf{P}(\boldsymbol{\eta}) = \prod_{q=1}^{Q-1} \pi_{\eta_q}$$

and $\mathsf{P}(j|\boldsymbol{\eta}, i)$ is the probability that a UED's queue length becomes j when the UED is in state i and the states of the other $Q-1$ UEDs are $\boldsymbol{\eta}$.

Assume ties are broken uniformly at random. If $\sum_{q=1}^{Q-1} \mathbf{1}_{\eta_q \leq i-1} \geq 1$, then

$$\mathsf{P}(j|\boldsymbol{\eta}, i) = \begin{cases} 1 & \text{if } j = i, \\ 0 & \text{if } j \neq i \end{cases}$$

because the tasks will be dispatched to UEDs, original queue lengths of which are smaller than i. On the other hand, if $\sum_{q=1}^{Q-1} \mathbf{1}_{\eta_q \leq i-1} < 1$, then the UED with queue length i will receive N tasks, and $\mathsf{P}(j|\boldsymbol{\eta}, i) = 1$ for $j = i + N$.

WLOS, we assume UED_Q has queue size i. Given any $j \geq 0$, we define $T_j = \sum_{q=1}^{Q-1} \mathbf{1}_{\eta_q = j}$, which is the number of UEDs with queue length j excluding UED_Q. T_j is then the sum of $Q-1$ i.i.d. Bernoulli r.v.'s with mean π_j; thus, $\mathsf{E}\, T_j = (Q-1)\pi_j$. Now, the probability that UED_Q receives N tasks is given by

$$\mathsf{E}\left(\frac{1 - \sum_{j=0}^{i-1} T_j}{1 + \sum_{j=0}^{i} T_j}\right)^+,$$

which, at steady state, can be approximated by

$$\left(\frac{1 - (Q-1)\sum_{j=0}^{i-1} \pi_j}{1 + (Q-1)\sum_{j=0}^{i} \pi_j}\right)^+$$

because T_j converges to $(Q-1)\pi_j$ in distribution and the term inside the expectation is bounded and continuous in terms of T_j. This concludes the proof. ■

According to Lemma 2, the queue length dynamic of a single UED can be represented by the Markov chain in Figure A.1. Intuitively, $\tau_{\boldsymbol{\pi}}$ indicates the queue length so that the probability that a UED with queue size $i(\geq \tau_{\boldsymbol{\pi}})$ receives N tasks is 0. Based on Lemma 2, we can calculate the stationary distribution of the queue length of a single UED numerically by finding $\hat{\pi}$ that satisfies the global balance equation.

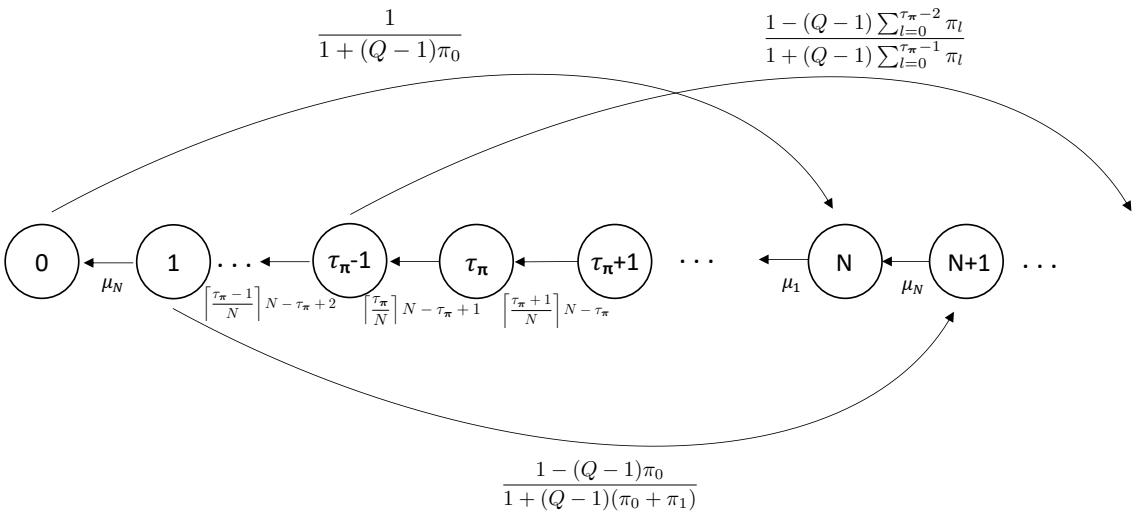


Fig. A.1.: The Markov chain representing the system

Lemma 3 *The expected service time* $T_Q(\lambda, \mu_1, \cdots, \mu_N)$ *of an application instance that is dispatched to* Q *UEDs is given by*

$$\sum_{r=0}^{N-1}\left[\sum_{i=1}^{\infty}\left(\lfloor\frac{i-1}{N}\rfloor\sum_{l=1}^{N}\frac{1}{\mu_l}+\mathbf{1}_{\lceil\frac{i}{N}\rceil N-i+1-r\geq 1}\sum_{m=\lceil\frac{i}{N}\rceil N-i+1-r}^{N-r}\frac{1}{\mu_m}+\right.\right.$$
$$\left.\left.\mathbf{1}_{\lceil\frac{i}{N}\rceil N-i+1-r<1}\sum_{m=N-r}^{\lceil\frac{i}{N}\rceil N-i+1-r+N}\frac{1}{\mu_m}\right)\cdot\left\{\left(\sum_{j=i-1}^{\infty}\pi_j\right)^Q-\left(\sum_{j=i}^{\infty}\pi_j\right)^Q\right\}\right].$$

Proof Task of type N becomes the ith task in the queue with probability $\left(\sum_{j=i-1}^{\infty}\pi_j(t)\right)^Q - \left(\sum_{j=i}^{\infty}\pi_j(t)\right)^Q$. Thus, the expected time a task spends in the system under our dispatch solution is

$$\sum_{i=1}^{\infty}\left(\lfloor\frac{i-1}{N}\rfloor\sum_{l=1}^{N}\frac{1}{\mu_l}+\sum_{m=\lceil\frac{i}{N}\rceil N-i+1}^{N}\frac{1}{\mu_m}\right)\cdot\left\{\left(\sum_{j=i-1}^{\infty}\pi_j\right)^Q-\left(\sum_{j=i}^{\infty}\pi_j\right)^Q\right\}.$$

For other type $N-r$ of a task, the cyclic structure in queue should be taken into account and there by changes an expression for the summation $\sum_{m=\lceil\frac{i}{N}\rceil N-i+1}^{N}\frac{1}{\mu_m}$, which leads to the desired result. ■