MOBILE DEVICE CLUSTERS AS EDGE COMPUTE RESOURCES: DESIGN, DEPLOYMENT, AND ROLE IN THE COMPUTING ECOSYSTEM

A Thesis
Presented to
The Academic Faculty

by

Karim Habak

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computer Science

School of Computer Science
Georgia Institute of Technology
August 2018

MOBILE DEVICE CLUSTERS AS EDGE COMPUTE RESOURCES: DESIGN, DEPLOYMENT, AND ROLE IN THE COMPUTING ECOSYSTEM

Approved by:

Professor Mostafa H. Ammar,
Co-advisor
School of Computer Science
Georgia Institute of Technology

Professor Ellen W. Zegura,
Co-advisor
School of Computer Science
Georgia Institute of Technology

Professor Ada Gavrilovska
School of Computer Science
Georgia Institute of Technology

Professor Umakishore Ramachandran
School of Computer Science
Georgia Institute of Technology

Professor Khaled Harras
School of Computer Science
Carnegie Mellon University Qatar

Date Approved: May 17, 2018

To my parents, for they unconditional love and support

To my wife and my daughter who shared the whole journey with me and added a huge amount of love and joy to it.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisors Prof. Mostafa Ammar and Prof. Ellen W. Zegura for their dedication, availability, guidance, patience, and support. Throughout my PhD journey, they gave me enough freedom to explore the topics I found exciting and they have always been there to provide guidance whenever I needed it. They significantly contributed to my academic development and guided me along the way of becoming an independent researcher. Their stimulating suggestions have helped me identify my strengths, overcome my weaknesses, shape my research skills. On a more personal note, they showed me how to lead by example, became among those who I am always looking up to, and significantly contributed to my personal development. To summaries, I cannot imagine completing my PhD journey without their continuous support.

Looking back at the last 10 years, Prof. Khaled A. Harras stands out among those who have the most influence on my career and personal life. Despite sharing the same continent for just around six month in aggregate, he has always demonstrated extreme levels of availability, support, and compassion. I truly understand and appreciate the efforts he puts in being available and aware of the details in spite of the distance and the difference of time zones. I am deeply indebted to Prof. Harras specially when it comes to my early career and personal development. On a more academic level, Prof. Harras has always been a truly fantastic mentor/advisor that I am fortunate to know and work with. His deep insights have significantly helped in shaping my work and writing about it. His passion working with students and young researchers makes him an amazing person to work with and learn from.

I am also grateful to the rest of my committee members: Prof. Umakishore Ramachandran and Prof. Ada Gavrilovska for their helpful comments and insights that enabled me to improve the work in this thesis. I owe many thanks to Prof. Constantine Dovrolis, and Prof Jim Xu for their discussions, encouragement, and suggestions on my research and career. I would also want to thank Dr. Shruti Sanadhya from HP Labs, Dave Oran from Cisco, and Dr. T. V. Lakshman, Dr. Sarit Mukherjee, and Dr. Fang Hao from Nokia Bell Labs for hosting me as a summer intern where I had the chance to work on a set of very interesting research problems that influenced the work presented in this thesis.

I would also like to thank a group of wonderful colleges in the Networking Research Group at Georgia Tech. Specially, Dr. Cong Shi, Dr. Ahmed Mansy, Dr. Samantha Lo, Dr. AliReza Khoshgoftar Monfared, Dr. Hyojoon Kim, Dr. Srikanth Sundaresan, Dr. Robert Lychev, Dr. Sam Burnett, Dr. Bilal Anwer, Dr. Yogesh Mundada, Dr. Maria Konte, Dr. Abhinav Narain, Dr. Aemen Lodhi, Dr. Saamer Akhshabi, Dr. Ilias Foudalis, Tarun Mangla, Yimeng Zhao, Sean Donovan, Sathya Gunasekaran, Kamal Shadi, Kaesar Sabrin, Payam Siyari, and Danny Lee for their support, encouragement, fruitful discussions and above all friendship.

A special thanks goes to one of my closest friend, Ahmed Saeed, who shared with me the last 13 years (and counting) of my life and my whole career so far. We started our research journey together and he has always been there in all the major events in my life and he was always present whenever I needed him. Our continuous discussions about work and other aspects of life are always helpful, stimulating, and motivating. Over the last few years, he and his wife, Heba Kamal, have always been like a family for me and my wife. We share with them lots of amazing memories that will never be forgotten. Mentioning family, I can't forget our American family, Dave Savage and Beverly Molander. They made our move to the United States easy. They were very welcoming, supportive, motivating, and above all loving and caring. My wife and I

cannot find better ways to describe their rule and impact in our lives other than truly being a family for us. It is best depicted by our daughter calling them ”grandma Beverly” and ”grandpa Dave”.

Last but not least, I would like to thank my parents, my brother Ahmed and my sister Mai for their unconditional love and support. Their rule in my life is beyond any description. Also nothing can describe how thankful I am to my wife, Sara, for her love, support, and patience. I would also love to thank my little daughter, Yara, who brought loads of joy and happiness to our lives. Finally, I cannot forget to thank my mother in law for her love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Edge computing offers an alternative to centralized, in-the-cloud compute services. Among the potential advantages of edge-computing are lower latency that improves responsiveness, reduced wide-area network congestion, and possibly greater privacy by keeping data more local. However, widely deploying the needed edge-compute resources requires (1) provisioning the load introduced at various locations, (2) huge initial deployment cost and management expenses, and (3) continuous upgrades to keep up with the increase in demand. The availability of under-utilized mobile and personal computing devices at the edge provides a potential solution to these deployment challenges. In this thesis, we propose taking advantage of clusters of co-located mobile devices to offer an edge computing platform. Scenarios with co-located devices include, but are not limited to, passengers with mobile devices using public transit services, students in classrooms and groups of people sitting in a coffee shop. We propose, design, implement and evaluate the Femtocloud system which provides a dynamic, self-configuring and multi-device mobile cloud out of a cluster of mobile devices. Within the Femtocloud system, we develop a variety of adaptive mechanisms and algorithms to manage the workload on the edge-resources and effectively mask their churn. These mechanisms enable building a reliable and efficient edge computing service on top of unreliable, voluntary resources. Our work also includes building a system that enable mobile devices to accurately and efficiently acquire knowledge of the existing compute service providers, their compute capacities, and the network parameters while communicating with each of these providers. Such data is acquired through measurements that involve a set of voluntary mobile devices and is be used to allow allow mobile devices to select the compute service provider

that matches their demand and meets their target level of quality of experience. The data acquired by our system can also be used by compute service providers to identify potential locations for service deployment and discover any shortcomings in their existing deployments.

CHAPTER I

INTRODUCTION

Since the 2002 paper by Balan et al. making a case for mobile devices to cyberforage by finding surrogate (i.e., helper) servers in the environment [9] the research community has explored various forms of interaction between mobile devices and fixed, higher capacity infrastructure, including the cloud. The motivation for this exploration has been and remains as articulated by Satyanarayanan[49], namely that mobile devices are resource constrained in comparison with servers, and that users desire high performance applications regardless of the device used to experience the application. By offloading some computation to more powerful servers, mobile devices can offer a user experience beyond what local capabilities can support. Further, offloading may allow mobile devices to save power and extend time between charges.

In addition to questions of performance speedup, energy savings and cost, the key questions for an offloading system design are: where is the higher performance capacity, who provides it, and how does it fit into a larger computing ecosystem? In traditional cloud computing, the higher performance capacity is located in data centers reached via the Internet and provided by companies that charge for transient server use. Traditional cloud computing can offer essentially unlimited compute capacity, but at the price of latency and bandwidth limitations between the mobile device and the servers in large data centers. In response to these limitations, the Cloudlet system moves computation closer to mobile devices, creating a two-tier architecture where a mobile device can offload to a nearby, less capable server, at low latency and high bandwidth, rather than (or as a complement to) offloading to the cloud [48]. In the cloudlet vision, these nearby servers would be located in public and

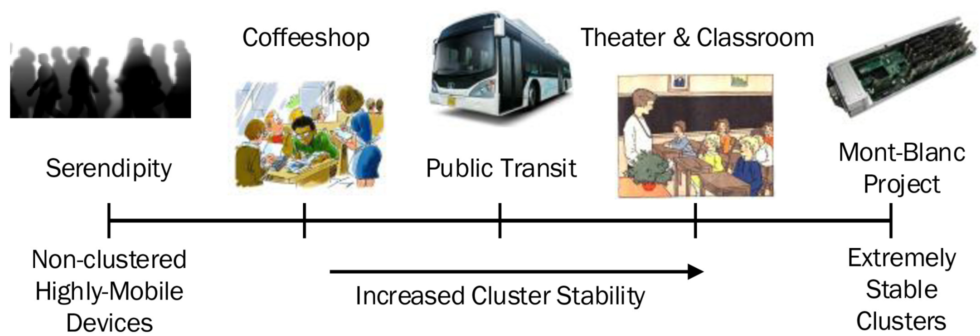


Figure 1: Mobile cluster stability spectrum.

commercial spaces where people congregate, such as coffee shops and airport waiting areas [48].

Although one could imagine a third-party provider owning and operating these cloudlets for profit, truly realizing the cloudlet's vision faces a set of deployment challenges. First, for every given location, capacity provisioning is needed to ensure meeting the requirements of the users in this location at any point in time. Second, the cost of covering the edge with dedicated computing-servers with such provisioned capacity is extremely high. Finally, the deployed infrastructure will need to be actively managed continuously upgraded to account for the changes in users-demand. These challenges form a formidable deployment barrier for cloudlet-like systems despite their clear advantages.

The availability of mobile and personal computing devices at the edge provide a potential solution to the cloudlet's deployment challenges. The fact that these devices are widely adopted and are often under-utilized suggests that there might be enough resources at the edge to operate an edge computing service without the need for deploying additional resources. To this end, we propose taking advantage of clusters of co-located mobile devices and build "femtoclouds" that offer an edge computing platform. We can situate our work relative to other approaches that use mobile devices to provide a compute service, by examining the stability and predictability along a spectrum as shown in Figure 1. At one end of the spectrum are extremely stable and deliberately configured clusters such as proposed in the Mont-Blanc project (www.montblanc-project.eu) where, motivated by energy considerations, a large number of mobile CPUs are configured in a single chassis. Our settings of interest – coffee shops, public transit systems and theaters/classrooms – fall in the middle of the spectrum with different levels of stability. At the other end of the spectrum are highly mobile and unpredictable devices that are used opportunistically as they are encountered over time (e.g., Serendipity [54]).

We envision a fully-operational compute service provider that relies primarily on clusters of voluntary mobile and edge devices to provide the needed compute resources. To this extent, the main statement in this thesis is: ***With appropriate management, mobile device clusters can provide useful and reliable edge compute resources despite their churn. These clusters can play a major role and fill in existing gaps in the evolving compute ecosystem***. In this thesis, we investigate the possibility of realizing our vision and take a set of key steps towards achieving it. To this end, this thesis consists of the following components:

The FemtoCloud System. The first key step towards realizing our vision is to examine the possibility of orchestrating a collection of co-located mobile devices to provide an edge computing resource. Specifically, we investigate the scenario where a collection of mobile devices, with shared compute resources and different availability periods, exist at some location (e.g., a classroom or a coffee shop). Our main objective is how to form a single compute resource out of these devices and maximize its computational throughput. To this end, we propose, design, and implement the FemtoCloud system that is able to cluster these mobile devices, estimate their shared compute capacity and availability durations, and use the shared capacity to offer an accessible computing platform. Within the FemtoCloud system, we formulate the task assignment and scheduling problem that strives to maximize the computational throughput of the underlying mobile device cluster. Furthermore, we develop a set of heuristics to efficiently solve our scheduling problem. Finally, we build a prototype of the system and use it in addition to simulations to evaluate its performance.

Workload Management in Edge Femtoclouds. The natural second step towards our vision is to extend our architecture to go beyond a single mobile device cluster and provide a service to job originators that is comparable to that provided by a centralized cloud service, namely the submission of jobs for completion in a timely and reliable manner. To achieve these goals, we extend the FemtoCloud system architecture to be more accessible by mobile devices sharing their resources as well as job originators. We also introduce a set of more realistic job model and user presence time model. In addition, we identify the importance of workload management for providing a reliable service to job originators. Therefore, we design a set of workload management mechanisms to efficiently manage the available resources and effectively mask the impact of churn. We implement a system prototype and use it, in addition to simulations, to evaluate the performance of the system and assess the efficiency of each of our developed mechanisms.

Characterizing and Navigating the Compute Ecosystem. Finally, a successful compute service provider should be able to coexist with the currently operating compute service providers. All these providers are key components of large compute ecosystem where users are able to select which compute service provider to use at any point in time. Therefore, we use measurements to shed light on the current compute ecosystem, highlight its complexity, and identify the best way to model its components. In addition, we develop mechanisms that help users to adaptively select the best compute service provider that matches their needs at any point in time.

1.1 Thesis Contributions

In this thesis, we make the following contributions:

1. **The FemtoCloud System**

   (a) We design, implement, and evaluate the FemtoCloud system that leverages the available compute resources on a cluster of mobile devices to offer compute resource at the edge.

   (b) We formulate the FemtoCloud task scheduling problem and develop efficient heuristics to solve it.

2. **Workload Management in Edge Femtoclouds**

   (a) We design a hybrid edge-cloud architecture that utilizes the cloud for management and to provide a stable service interface while using the edge for low latency computation.

   (b) We develop a set of workload management mechanisms that enable an edge computing service comprised of mobile devices with churn to serve directed acyclic graph (DAG) structured jobs.

   (c) We implement a prototype of our system that we use to evaluate the performance of the Femtocloud system. We also use simulations to assess the efficiency of each of our workload management mechanisms, independently.

   (d) We perform a pilot study to identify suitable incentive mechanisms to encourage users to opt in a Femtocloud system and share their mobile compute resources.

3. **Characterizing and Navigating the Compute Ecosystem**

   (a) We present a measurement study that characterizes the current state of the compute ecosystem and identify suitable models to abstract its components.

(b) We design a system and proposed mechanisms that allow mobile devices to efficiently navigate the increasingly complex compute ecosystem and efficiently select the compute service provider that matches their needs.

(c) We implement a prototype of our system that we use to evaluate its performance in a controlled settings and in the wild.

1.2 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 presents the current state-of-the-art and discusses the work related to this thesis. In Chapter 3, we investigate how to form an edge compute resource using a group of co-located voluntary mobile devices with churn. Chapter 4 presents a set of workload management functions that enables us to provide a reliable and efficient edge computing service on top of unreliable, voluntary resources. We present a measurement study to better understand the computing ecosystem in Chapter 5. Chapter 6 concludes this thesis, summarizes the main contributions, and discusses potential directions for future work.

CHAPTER II

RELATED WORK

This chapter provides an overview of the work related to this thesis. We start by presenting the cloud-based computational offloading work followed by the state-of-the-art in the area of edge computing. We then shed the light on a set of key mechanisms used for workload management in traditional data centers. Finally, we present an overview of the techniques used for network parameters measurement and estimation.

2.1 Cloud-Based Computational Offloading

Computational offloading has emerged as a result of the lack of computational resources in the early generations of smart-phones, the limited battery these devices operate-on, and emerging trends of developing compute intensive applications [21]. Early computational offloading systems argued for offloading the heavy computations from the resource-constrained mobile devices to the cloud, which has virtually unlimited compute capacity [53, 19, 17, 34, 29, 56]. For instance, MAUI [19] and CloneCloud [17] primarily focused on enabling code offloading without placing too much burden on the application developers. COSMOS [53], however, focused on building a cost-efficient computational offloading service on top of commercially-available, on-demand, elastic clouds. These systems demonstrated combined energy-savings and execution speedups for applications with appropriate properties. In contrast, these systems also showed that the communication bandwidth and the latency between the mobile device and the cloud are the main performance bottlenecks for the cloud-based computational offloading systems. In addition, they demonstrated that some applications can tolerate higher communication delays between the mobile device and the cloud than others.

2.2 Edge Computing

To minimize the network delays between the mobile devices and the computing servers, edge computing proposes bringing compute resources closer to their users. One early realization of edge computing was Cloudlets, which introduced a middle tier between mobile devices and traditional clouds [48, 30, 68, 37]. Cisco's fog computing also shares the same vision of introducing a layer between data centers and end devices [11, 59, 58, 12]. The end goal of both Cloudlets and fog computing is covering the edge with dedicated computing servers. Therefore, they share the following key limitations: (1) They need capacity provisioning to ensure meeting the requirements of the users in a given location at any point in time; (2) Their initial deployment cost is extremely high due to the need to cover the edge with dedicated servers; (3) The deployed infrastructure will need to be actively managed and continuously upgraded to account for changes in user demands. These limitations stand as a formidable barrier that hinders the deployment of either of these systems despite their clear performance advantages.

To reduce the deployment cost and overhead of edge computing services, recent efforts proposed leveraging the resources of underutilized mobile devices at the edge [44, 15, 54, 40, 22, 47, 25, 31, 7]. These approaches, however, focus on one of two extreme scenarios. First, scenarios with extremely high device mobility have been thoroughly investigated [54, 55, 41]. In these scenarios, forming clusters out of these mobile device is relatively impractical due their high mobility. Therefore, these devices can only be used in an opportunistic manner. The second class of scenarios suggests that the devices are expected to be available at certain location for extended periods of time. Therefore, they can be deliberately configured to form a stable computing cluster [40]. Our work in this thesis fills the gap between these two extremes. We uniquely propose taking advantage of clusters of devices that tend to be co-located in places such as public transit, classrooms, theaters, or coffee shops.

These locations have some elements of social and/or physical structure that suggest the ability to predict the device's availability durations. This information can be taken into account to build a reliable and efficient edge computing service on top of unreliable, voluntary resources.

2.3 Workload Management

Workload management plays a major role in any distributed computing system [67, 65]. It consists of a set of functions that optimize the performance of the system by (1) controlling the admission of new jobs to avoid overwhelming the available resources [66], (2) allocating enough resources for the admitted jobs [63, 10], (3) distributing the work on the available compute resources [61, 35, 57], (4) balancing the load on the existing resources [20, 39], and (3) checkpointing the work in progress to tolerate faults [64, 5, 23, 14]. Despite the fact that workload management has been thoroughly studied in the last two decades, there are two fundamental differences in our environment that require innovations within the femtocloud scope beyond existing research in workload management: (1) the fragmented and heterogeneous nature of the compute resources provided by mobile devices, and (2) the potentially significant but predictable churn in the availability of this compute resource. In this thesis, we build a set of workload management functions are carefully designed for femtocloud scenarios. We demonstrate that taking the predictability churn into account leads to the design of efficient workload management functions that can mitigate the impact of high churn.

2.4 Network Measurements

Over the years, network measurement has been an actively studied research field. Two of the heavily studied research directions in this field are (1) network parameters estimation, and (2) network topology mapping. In network parameters estimation, the main goal is to develop tools and techniques to estimate the communication parameters (e.g., bandwidth, latency, round-trip time) certain network routes that connect two or more nodes [45, 60, 43, 32, 50]. In the network topology mapping, however, the main goal is to have the current view of the network topology that connects certain entities [6, 16, 38, 18].

To understand the role played by FemtoCloud-like systems in the continuously evolving compute ecosystem, we need to first have the current view of the network topology that connects edge devices to the cloud. Therefore, we conduct a measurement study that to understand the network topology and connectivity between a set of geographically distributed edge devices and different cloud data centers. The measurements in this thesis belongs to the network topology mapping line of work and relies on state-of-the-art network measurement tools. In addition, we rely on our acquired measurements to better model the different compute options in the compute ecosystem and then develop techniques for selecting which compute option to use.

CHAPTER III

THE FEMTOCLOUD SYSTEM

3.1 Introduction

This chapter examines the possibility of orchestrating a collection of co-located mobile devices to provide a viable compute resource at the edge. We focus on scenarios of co-located devices that have some elements of social and/or physical structure. This structure suggests forms of stability and, more importantly, predictability of the duration of time in which a given mobile device is available to be used as a part of this compute resource. We use the term "femtocloud" to refer the collection of mobile devices that are configured into an edge compute resource.

Fortunately, the scenarios that have the needed social and/or physical structure are very common. For instance, these scenarios include, but are not limited to, passengers with mobile devices using public transit services, students in classrooms, and groups of people sitting in a coffee shop. In addition, these scenarios share a set of properties that help in guiding the design our FemtoCloud System:

- There is a natural owner of the setting who may have a business interest in providing (or contracting for) a controller that puts this femtocloud together and makes it work, whether a coffee shop owner, a university, a theater owner, or a public transport provider.

- Each setting has semantics that suggest forms of stability and, importantly, predictability in the duration of time that a given device is available for use in the femtocloud. A classroom has pre-determined time periods of use – during a scheduled class – and predetermined times when the occupancy will experience most turnover. Public transportation offers only fixed chances for occupancy change, based on bus or train stops. A coffee shop is more complicated from a stability and predictability standpoint; for now, we simply observe that coffee shop patrons fall into at least two classes – those who stand in line, make a purchase, and leave immediately, and those who linger.

- There is a potential to build trust based on in-person, social relationships. Coffee shop patrons, students at a university, and public transport riders are typically repeat customers with a relationship of some form to the owner of the setting. Repeat participation also provides the potential to learn about devices and device owners in ways that can optimize femtocloud usage.

- There is a natural form of payment to those who participate, associated with the setting, such as coffee shop credit, university currency credit, or public transportation credit. While other forms of compensation for device use are certainly possible, these options connect the place where the device is used to the compensation in ways that may be attractive to device owners and the setting owner.

In this chapter, we propose the FemtoCloud system which provides a dynamic, self-configuring and multi-device mobile cloud out of a cluster of mobile devices that exists in one of our scenarios of interest. We present the FemtoCloud system architecture designed to enable multiple mobile devices to be configured into useful edge compute resources despite churn in mobile device participation. Our architecture consists of two main components: A controller that is deployed by the owner of the setting, and a collection of mobile devices that dynamically come into and leave our setting. We formulate the FemtoCloud task assignment and scheduling problem as a mixed integer linear programing problem that takes into account the devices' availability durations, their capacities and the requirements of the tasks in order to take an optimal assignment decision. Furthermore, we develop a set of heuristics to efficiently reach an approximate solution of this optimization problem. We build a prototype of our FemtoCloud system and use it in addition to simulations to evaluate the performance of the system showing its efficiency and ability to leverage the available compute capacity of the volunteering devices.

The remainder of this chapter is organized as follows. We begin in Section 3.2 with the architecture, identifying functionality to be realized in the controller and in the mobile devices, and information to communicate within and between the two. We identify a critical and obvious problem that must be solved at the controller, namely the scheduling of tasks onto mobile devices where the transmission of data and receipt of results all happens over a shared wireless channel. We formalize the scheduling problem and then develop several algorithms in Section 3.3. We evaluate the system in Section 3.4 using simulations, including those driven by measurements of device dynamics in different settings. We also describe and report briefly on a prototype built on Android. We end the chapter with a Summary in Section 3.5.

3.2 The FemtoCloud System

The FemtoCloud computing service executes a variety of tasks that arrive at the control device. The FemtoCloud client service, running on the mobile devices, estimates the computational capability of the mobile device, and uses it along with user input to determine the computational capacity available for sharing. This client leverages device sensors, user input, and utilization history, to build and maintain a user profile. Afterwards, the service shares the available information with the control device, which is then responsible for estimating the user presence time and configuring the participating mobile devices as a cloud offering compute as a service.

In this section, we present the details of the FemtoCloud system. We start with listing our assumptions followed by the detailed description of our architecture depicted in Figure 2. Afterwards, we present the implementation details of our prototype.

3.2.1 Assumptions

We assume that some users will have the FemtoCloud client service installed on their mobile devices, and that they are willing to share a portion of their computational capabilities as a result of different incentives ranging from their willingness to share resources (as in SETI or BOINC) to direct financial gains. We acknowledge that such incentive mechanism is essential specially for users with battery operated devices. We assume that a femtocloud controller is responsible for deciding which mobile devices will be added to the compute cluster in the current environment.

We assume a general task arrival model where tasks can arrive individually or in batches following any task arrival distribution. Each of these tasks is a *compute intensive* tasks that has its own computation requirements, input data size, and output data size. We assume that a task assigned to a mobile device needs to be completed and the results returned to the control device before the mobile device leaves the cluster. Otherwise, the task is aborted and may need to be re-assigned and restarted. Based on these task parameters as well as the availability of mobile devices, the controller builds a task execution schedule and assigns each task to a mobile device to optimize the metric of interest.

3.2.2 System Architecture

The mobile device functions are performed in the following modules:

User Interface Module: This module obtains user preferences, resource sharing policies and personal profile sharing policy. For instance, the user can configure this module to share up to certain percentage of his mobile device capabilities, or contribute to femtocloud only if the battery level is above certain threshold. They

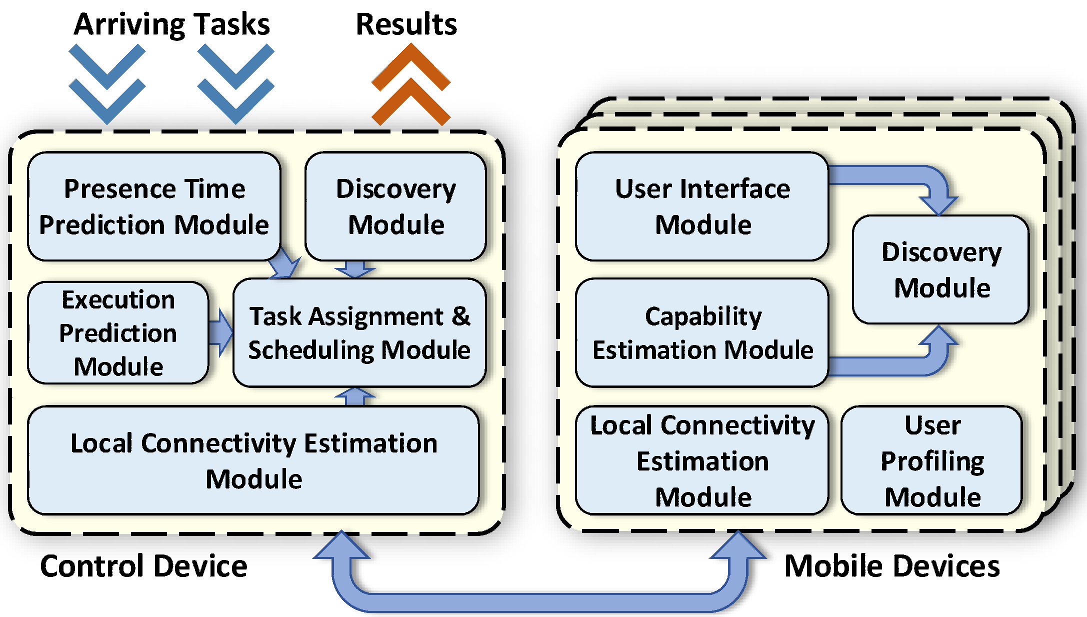


Figure 2: The FemtoCloud system architecture

also define policies that dictates whether they are willing to join a femtocloud or not. These policies are defined by many factors such as available battery level, time, environment type, etc.

Capability Estimation Module: This module estimates the computational capabilities of the mobile device including the number of cores in the device and the available computational capacity. Such computation capacity varies based on the system load and whether the device running a power savings mode. The computational capacity estimate is shared with the control device. Since the estimate may change over time, this module periodically sends updated estimates to the control device.

User Profiling Module: This module gather data about the user preferences and behavior in different scenarios to be used for determining his presence time while joining femtocloud. This module opportunistically mines the gathered data and build a profile. This module only share the profile with the controller in user accepted granularity to maintain user privacy.

The control device functions are performed by the following modules:

Execution Prediction Module: In order to efficiently distribute tasks across different processing nodes, the controller should know the execution load introduced by each of these tasks. To achieve this goal, we rely on the original task source to provide the controller with this information. However, if the source does not provide such information, the control device carries the responsibility of connecting to an execution estimation service to acquire it. Such estimation may be done using the Mantis system [36].

Presence Time Prediction Module: This module is responsible for predicting the presence time for femtocloud users. It gathers environment specific data to build a generic user profile based on the collective behaviors of the users. This profile is used to estimate the presence time for new users as well as updating the estimates over time. It also uses specific user profile, if shared, along with this generic profile to determines his presence time.

Task Assignment and Scheduling Module: This module uses the information acquired by the previous modules to iteratively assign tasks to their executing devices.

The control device collaborates with the mobile devices in the cluster to perform functions implemented in the following modules which are instantiated in both types of device.

Local Connectivity Estimation Module: This module estimates the available bandwidth between the control device and each computing mobile device. Since these devices are directly connected and relatively static in most of the scenarios, many techniques can be used to estimate the available bandwidth. Our approach is to use the wireless signal strength to get the initial estimate of the bandwidth and then monitor the actual achievable bandwidth while assigning tasks and/or gathering results to update such estimate.

Discovery Module: This module discovers the available mobile devices that have the FemtoCloud client service installed. Once a mobile device becomes ready to join a cluster, it sends a registration packet to the control device. This registration packet can includes an initial estimate of the compute capacity based previous contribution to the FemtoCloud system in similar context and user profile information to be used for determining his presence time. We also use periodic heartbeats to keep track of the devices in the cluster and gather more updated information about their shared computational capacity.

3.2.3 Implementation

To assess the feasibility of FemtoCloud and evaluate it, we implement a FemtoCloud prototype in Android.

We implement the control device to hold the responsibility of providing an interface to the task originators and to manage the mobile devices inside the cloud. The interface to the task originators enables them to send the code for the desired computation coupled with their input data and to receive the results once they become available. The control device works in collaboration with the FemtoCloud client service installed in the mobile devices to acquire information about the device characteristics and user profiles. The service uses such information to assign task to devices according to a heuristic which will be described in the next section. To minimize the communication overhead and enhance performance we first use persistent TCP connections between each mobile device and the control device to avoid the delays introduced by the protocol's handshaking and slow start mechanisms. We also allow the control device to act as a WiFi hotspot allowing the mobile devices to connect to it using infrastructure mode.

Note that there is no contention for the communication channel between the control device and the mobile devices in the cluster due to our scheduling technique. For communication from the mobile devices to the control device there will be two types: 1) short notifications and alerts that are allowed at any time and may contend for the channel, and 2) possibly longer communication needed to return computation results to the controller which are *scheduled* by the control device.

We implement the FemtoCloud client service as an Android application that allows the user to enter preferences and resource sharing policies. Once a user accepts to share a portion of the mobile device's resources and select the granularity of sharing his profile with the controller, it connects to the WiFi network offered by the controller. Upon successful connection, this service holds the responsibility of estimating the mobile device capabilities and sharing them with the controlling device. In addition, it works in collaboration with the control device to estimate the available bandwidth between the mobile device and the control device as well as the user presence time. More importantly, it carries the responsibility of executing tasks assigned to it by the controller. Upon completing the execution of a task, the client service stores the results, notifies the control device regarding the availability of such results, and starts executing other assigned tasks, if any. Finally, once it receives a request for the results from the controller, it sends the available results to the controller, deletes them and erases any stored state information about the task.

3.3 FemtoCloud Scheduling Problem

The scheduling algorithm that runs at the controller is critical to the performance of the system. The scheduler must assign tasks to available devices to maximize the metric of interest, while managing device churn. The task assignment problem differs from standard parallel task assignment because sending and receiving tasks takes place over a shared wireless channel and because we assume that if a device departs prior to completing and delivering its task result, the task must be reassigned and restarted from the beginning. These two constraints place a priority on getting tasks assigned quickly, executed well within estimates of device persistence, and results returned quickly to the controller. While other metrics are possible, we focus on

Table 1: List of symbols used

Symbol	Description
$\mathcal{B}_k$	The available bandwidth at the k^{th} device
$\mathcal{C}_k$	The shared processing capacity of the k^{th} device
$\mathcal{T}_k^d$	The departure time of the k^{th} device
$\mathcal{E}_i$	The execution load of the i^{th} task
I_i	The size of the transferable input data and executable code of the i^{th} task
R_i	The size of the results of the i^{th} task
x_{ik}	Equals 1 if the i^{th} task is assigned to the k^{th} device and equals 0 otherwise
n	Number of tasks waiting for assignment
m	Number of devices in the cluster

maximizing the "useful computation", defined as total computation completed by the system.

We begin by formulating the problem as an optimization problem, assuming perfect knowledge of device capabilities (computation and bandwidth) and departure time. We then describe a greedy heuristic based on insights gained by solving the optimization problem on small instances.

3.3.1 Scheduling as Optimization

Table 1 summarizes the system parameters and notation used in the optimization. We assume the scheduler must distribute a batch of n tasks across the available mobile devices and gather their results. We assume that our cluster consists of m mobile devices with users willing to share their computation capabilities. Let $\mathcal{C}_k$ denote the shared computation capability of the k^{th} mobile device, $\mathcal{B}_k$ denote the available communication bandwidth between this device and the controller, and $\mathcal{T}_k^d$ denote the departure time of this device. For each of the n tasks, $\mathcal{E}_i$ denotes the execution load introduced by task i, I_i denotes the size of the code and its inputs, and R_i denotes the size of results of the same task.

Our goal is to determine a complete task-execution schedule. For each task, the scheduler determines which device to assign the task to, when to send the task to the device, and when to schedule the return of the result. The overall objective of the task-execution scheduler is to maximize the overall cluster's useful computations:

$$\text{Maximize } \mathcal{C} = \sum_i \mathcal{E}_i \sum_k x_{ik} \tag{1}$$

where $\mathcal{C}$ is the completed computational load and $\mathcal{E}_i$ is the execution load introduced by the i^{th} task.

The decision variables are: (1) $x_{ik}, \forall i, k$ where $x_{i,k}$ equals 1 if task i is assigned to device k and equals 0 otherwise. (2) The times of assigning a task to a processor, executing it, and sending its results to the controller. Therefore, solving this optimization produces not only a task assignment table but also a complete schedule for task transmission, execution and results transmission.

The following constraints must be satisfied: ***Integral Association:*** Each task should be assigned to at most one mobile device:

$$\sum_k x_{ik} \leq 1, \qquad \forall_i \tag{2}$$

Task Execution Schedule: For each assigned task, the elapsed time from the start of a task assignment to the start of its execution must be sufficient for a complete transmission of its code and data. In addition, the elapsed time from the start of executing a task to the start of sending its results back must be sufficient for completing its execution.

$$\left.\begin{aligned} \mathcal{T}_{E,i} - \left(\mathcal{T}_{S,i} + I_i \sum_k \frac{x_{ik}}{\mathcal{B}_k}\right) \geq M\left(\sum_k x_{ik} - 1\right), \\ \mathcal{T}_{R,i} - \left(\mathcal{T}_{E,i} + \mathcal{E}_i \sum_k \frac{x_{ik}}{\mathcal{C}_k}\right) \geq M\left(\sum_k x_{ik} - 1\right) \end{aligned}\right\} \qquad \forall_i \tag{3}$$

Where, for the i^{th} task: $\mathcal{T}_{E,i}$ is the starting time of its execution, $\mathcal{T}_{S,i}$ is the starting time of sending it to its executing processor, $\mathcal{T}_{R,i}$ is the starting time of sending its results, I_i is the size of the transferable input data and codes of the task and $\mathcal{C}_k$ is the processing capacity of the k^{th} processing node.

Processing Node Availability: For each assigned task, the controlling node must completely receive the results before arriving at the next station.

$$\mathfrak{T}_{\text{Limit}} - \left(\mathcal{T}_{R,i} + R_i \sum_k \frac{x_{ik}}{\mathcal{B}_k}\right) \geq M \left(\sum_k x_{ik} - 1\right), \qquad \forall i \tag{4}$$

Where $\mathfrak{T}_{\text{Limit}}$ is the time needed to reach the next station, $\mathcal{T}_{R,i}$ is the starting time of sending the results of task i, R_i is the size of the results of the i^{th} task, $\mathcal{B}_k$ is the communication bandwidth between the controller and processing node k, and M is a very large number used for constraint linearization. Basically, this constraint will become $\mathfrak{T}_{\text{Limit}} - \left(\mathcal{T}_{R,i} + \frac{R_i}{\mathcal{B}_k}\right) \geq 0$ if the i^{th} task is assigned to processing node k and become $\mathfrak{T}_{\text{Limit}} \geq -M$ if the task is not assigned to any processing node.

Wireless Channel Access Schedule: Since the processing nodes uses the shared wireless media to exchange data and results with the controlling node, overlapping the communication between different tasks will introduce delay for each of them. Such delay will lead to a decrease in our system performance. Therefore, one of our main design decisions is to avoid any overlapping communication between different tasks. To enforce such decision we use the following constraints:

$$\left.\begin{aligned}
\mathcal{T}_{S,i} + I_i \sum_k \frac{x_{ik}}{\mathcal{B}_k} - \mathcal{T}_{O,j} &\leq M\left(1 - z_{i^+j^+}\right), \\
\mathcal{T}_{S,i} + I_i \sum_k \frac{x_{ik}}{\mathcal{B}_k} - \mathcal{T}_{R,j} &\leq M\left(1 - z_{i^+j^-}\right), \\
\mathcal{T}_{R,i} + R_i \sum_k \frac{x_{ik}}{\mathcal{B}_k} - \mathcal{T}_{O,j} &\leq M\left(1 - z_{i^-j^+}\right), \\
\mathcal{T}_{R,i} + R_i \sum_k \frac{x_{ik}}{\mathcal{B}_k} - \mathcal{T}_{O,j} &\leq M\left(1 - z_{i^-j^-}\right),
\end{aligned}\right\} \qquad \forall_{i,j}, i \neq j \tag{5}$$

Where $z_{i^{\pm}j^{\pm}}$ reflects the order of media access operations, $+$ refers to the operation of sending a task to its executing processor, and $-$ refers to the operation of sending the results. For example, $z_{i^-j^+}$ equals 1 if the i^{th} task will finish sending its results before the j^{th} task starts its assignment process.

To maintain consistency and correctness while setting the values of $z_{i^\pm j^\pm}$, we use the following constraints:

$$\left.\begin{aligned} z_{i\pm j\pm} + z_{i\pm j\pm} - 1 &\leq M\left(1 - \sum_k x_{ik}\right), \\ z_{i\pm j\pm} + z_{i\pm j\pm} - 1 &\leq M\left(1 - \sum_k x_{jk}\right), \\ z_{i\pm j\pm} + z_{i\pm j\pm} + 1 &\geq \sum_k (x_{ik} + x_{jk}) \end{aligned}\right\} \qquad \forall_{i,j,i\neq j}$$

Processor Access Schedule: The following constraint is used to avoid overlapping the execution of tasks that are assigned to the same processor.

$$\mathcal{T}_{E,i} + \mathcal{E}_i \sum_k \frac{x_{ik}}{\mathcal{C}_k} - \mathcal{T}_{E,j} \leq M\left(1 - y_{ij}\right), \forall i, j \tag{6}$$

Where y_{ij} reflects the execution order of tasks i and j if they are assigned to the same processor. y_{ij} equals 1 if tasks i and j are assigned to the same processor and task i will be executed before task j. Since y_{ij} becomes one of the decision variables, we use the following constraint to insure that their values are consistent:

$$\begin{aligned} y_{ij} + y_{ji} &\leq 1, \qquad \forall_{i,j} \\ y_{ij} + y_{ji} &\geq x_{ik} + x_{jk} - 1, \qquad \forall_{i,j,k}, \qquad i \neq j \end{aligned}$$

Variable Ranges: The trivial constraints for the range of the decision variables are as follows:

$$x_{ik}, y_{ij}, z_{i^\pm j^\pm} \in \{0, 1\} \qquad \forall_{i,j,k}$$

$$0 \leq \qquad \mathcal{T}_{S,i}, \mathcal{T}_{E,i}, \mathcal{T}_{R,i} \qquad \leq \mathfrak{T}_{\text{Limit}}$$

Generally, this task assignment problem is a mixed 0-1 integer programming problem that can be shown to be NP-Complete. However, this problem definition guides us towards developing heuristics for task assignment and gathering of results.

3.3.2 Heuristics

Task Assignment Heuristics: To provide an efficient solution for our task assignment problem, we adopt an iterative greedy approach to assigning tasks to mobile devices. Our approach is based on three key ideas: (1) To maximize the efficiency of utilizing the communication channel, tasks with higher computational requirement per unit data transfer ($\frac{\mathcal{E}_i}{I_i+R_i}$) are prioritized. This approach increases the efficiency of using the mobile devices because it increases the probability of keeping device CPUs busy with tasks that require a lot of compute power while buffering new tasks at the controller. (2) To maximize the useful computation and increase processor utilization, the task is assigned to the mobile device that enables getting its results earlier regardless of the time taken to send the results of previously assigned tasks as long as (i) it will be able to send the results before leaving the cluster and (ii) it maintains the feasibility of receiving the results of the previously assigned tasks. (3) To maximize the amount of tasks executed by the cluster, the controller assigns as many tasks as it possibly can before a results gathering event is triggered by our *results gathering heuristics.*

Results Gathering Heuristics: Determining when to start gathering the available results from the devices is a very important question. Premature gathering of results wastes an opportunity of sending more tasks to the executing nodes and increasing computational throughput. Late gathering, however, risks wasting a portion of the results and having to reassign some incomplete tasks, which decreases the computational throughput. Therefore, we adopt two mechanisms while gathering results: (1) essential gathering mechanism and (2) early gathering mechanism.

The essential gathering mechanism clusters the results that have to be transmitted together and sends them in the following case:

$$\mathcal{T}_{\text{remaining}} < (1+\alpha)\mathcal{T}_{\text{needed}}$$

where $\mathcal{T}_{\text{remaining}}$ is the remaining time before the deadline, $\mathcal{T}_{\text{needed}}$ is the needed time to completely send these results to the controller, and α is a safety factor determined by the controller based on how accurate its estimates are about the available bandwidths and the departure times. We highlight that once the essential gathering event is triggered, we use "earliest deadline first" heuristic to gather the clustered results.

The early gathering mechanism utilizes the network in case of the absence of feasible assignment of new tasks to obtain the results. This approach keeps gathering one-task result at a time from the available results with the soonest deadline, until a new task arrival occurs or a change of the system status and parameters takes place.

3.4 Evaluation

In this section we evaluate the performance of the FemtoCloud system, We start by describing our experimental setup followed by presenting and analyzing our results.

3.4.1 Experimental Setup

To have a realistic performance evaluation, we start by identifying the available capacity in real mobile devices, and the compute requirements of real applications. First, we conduct a measurement study running a matrix multiplication application, we develop, with different preset computational loads (MFLOPs) on a set of mobile devices. We summarize the results of this study in in Table 2, which shows the average background thread capacity for the mobile devices. We conduct another measurement study to determine the compute resource usage of different real applications. Table 4 summarizes this study and shows the compute resource usage of the following three applications: (1) Chess game in high difficulty mode, (2) a video game called Angry Bird Space, and (3) Object recognition in a video feed (Video Processing). In our evaluation, we use the results of these studies coupled with a newly defined compute intensive application.

Tables 2, 4, and 3 summarize our experimental parameters. Throughout our

Table 2: Experimental Device's Characteristics.

Devices	Computation Capacity
Galaxy S5	3.3 MFLOPS
Nexus 7 [2012]	7.1 MFLOPS
Nexus 7 [2013]	8.5 MFLOPS
Nexus 10 [2013]	10.7 MFLOPS

Table 3: Experimental parameters. The underlined values are the defaults.

Parameter	Values
Chess input size (MBytes)	[0.5, <u>2</u>, 16]
Average user arrival rate (user/min)	[<u>2</u>, 8]
Average user presence time (min)	[0.25, <u>2</u>, 5]
Average device's available bandwidth (Mbps)	<u>20</u>
Average presence error ratio (%)	[-50, <u>0.0</u>, 50]

evaluation, we use a Poisson arrival process to model the arrival of new users as well as the arrival of new tasks. We use the following performance metrics:

- **Computational Throughput:** This is the average amount of useful computations finished by our femtocloud per second (MFLOPS).

- **Compute Resource Utilization:** This is the average utilization of the compute resources in our cluster. To calculate this utilization, we only consider useful computations, which belong to tasks completed by femtocloud.

- **Network utilization:** This is the average busy time of the network for sending tasks or receiving results.

Overall, we conduct two different sets of experiments. The first set of experiments (Section 3.4.2) aims for understanding the effect of different environmental parameters on the performance of femtocloud. In these experiments, we simulated different environments and studied the effect of different parameters in the performance of femtocloud. The second set of experiments (Section 3.4.3) sheds light on the performance of our developed prototype.

Table 4: Experimental tasks characteristics and evaluation parameters.

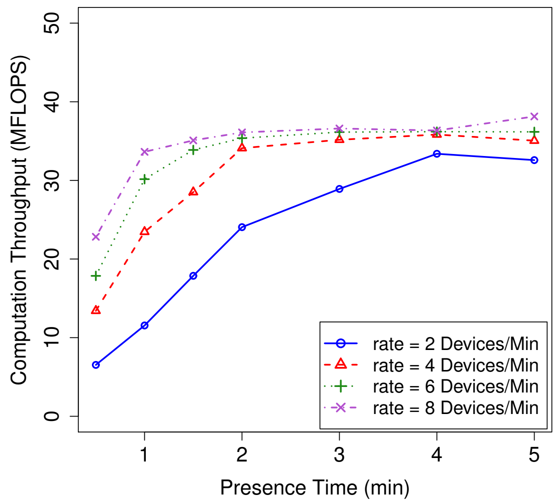
Task Type	Input	Computation	Output	arrival rate
Chess	2 MBytes	10 MFLOPs	0.2 MBytes	1 task/sec
Video Game	0.2 MBytes	30 MFLOPs	2 MBytes	2 task/sec
Video Processing	3.125 MBytes	60 MFLOPs	1 MBytes	1 task/sec
Compute Intensive	8 MBytes	100 MFLOPs	0.5 MBytes	0.5 task/sec

3.4.2 Femtocloud Simulation Results

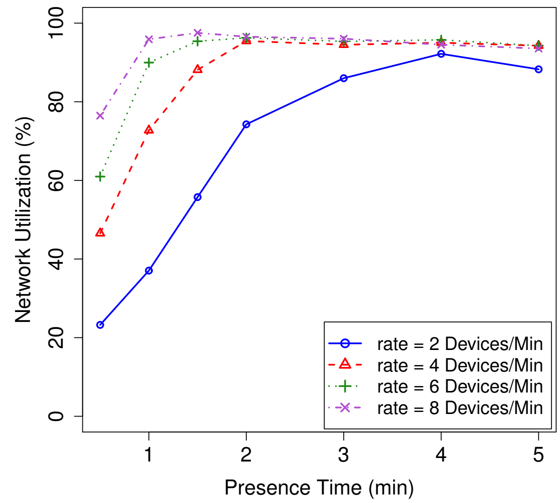
In this section, we study the impact of changing different environmental parameters on the performance of femtocloud. We start by studying the impact of user arrival rate and presence time followed by the true effect of stability in the system. We also study the impact of changing task characteristics and robustness to estimation errors. In a subset of these experiments, we compare femtocloud against a presence time oblivious scheduler (PreOb). Such scheduler uses the same task assignment heuristic used by femtocloud but without taking the presence time of a device into account. Due to its unawareness of the presence time, it requests the results from the device one they become available.

Impact of changing user arrival rate and presence time: Figure 3 shows the effect of changing the average user presence time and the average user arrival rate on the performance of femtocloud. Figure 3(a) shows that the increase in the presence time or the user arrival rate, significantly enhances the performance and increases the femtocloud's computational throughput. This increased computational throughput saturates for large values of the arrival rate or the presence time. To explain the reason behind this saturation, we refer to Figure 3(b), which clearly shows that the network utilization increases as more tasks get assigned to our devices until it becomes highly utilized and unable to support more task assignments.

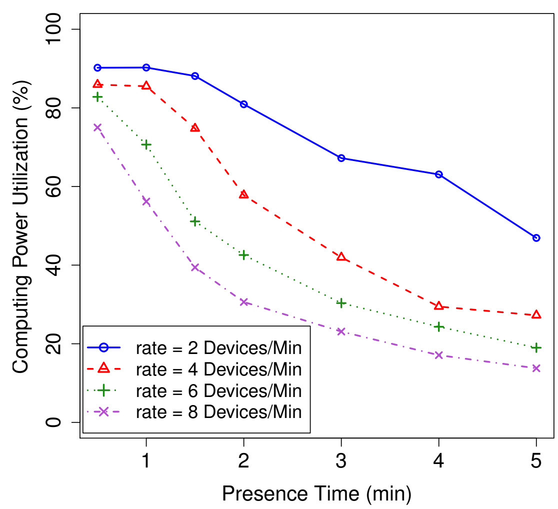
Figure 3(c) shows that the devices' utilization decreases with the increase of the presence time or the arrival rate. This decrease is due to having a lot of available devices in the system which enables distributing the load on them and minimizing



(a) Computational Throughput



(b) Network Utilization

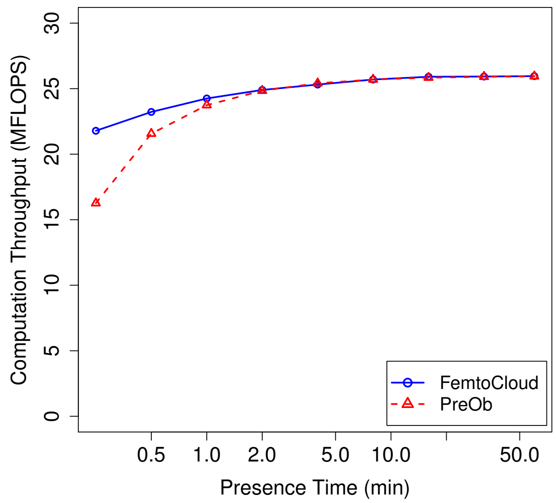


(c) Computational Resource Utilization

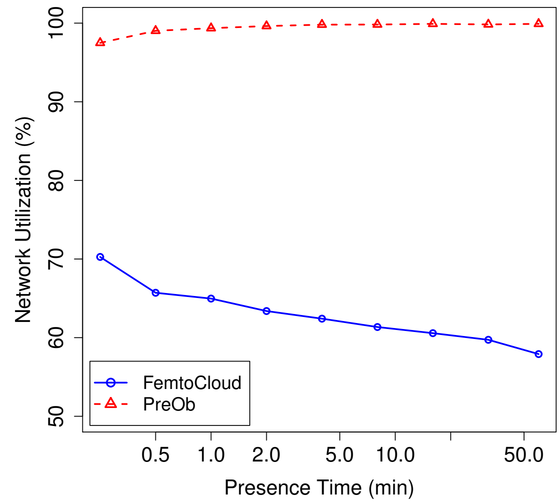
Figure 3: Impact of changing device arrival rate and presence time.

their utilization and overhead.

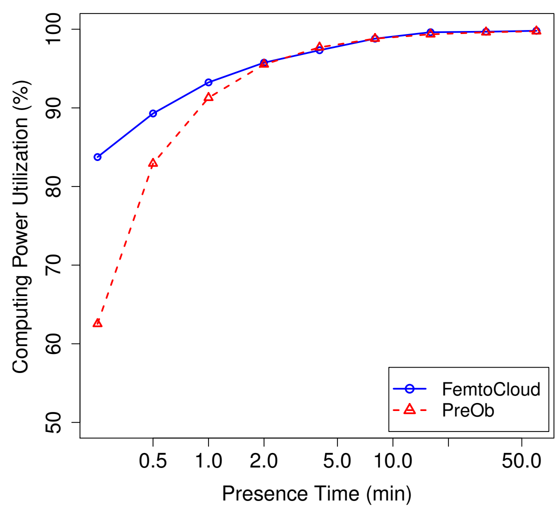
Stability Impact: Guided by the derivations we draw from Figure 3(c), it is critical to understand the true impact of the increased user presence on the system independent of the changes to the available compute resources. Therefore we construct an experiment in which we fix all the parameters in the system except the presence time



(a) Computational Throughput



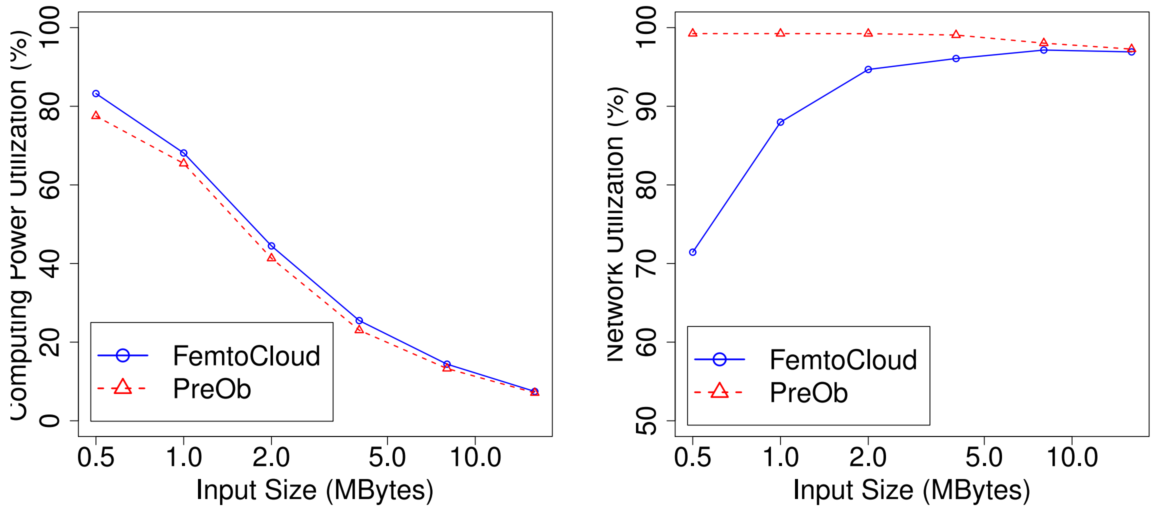
(b) Network Utilization



(c) Computational Resource Utilization

Figure 4: Stability impact.

of the devices. In this experiment we have three devices (a Nexus 10 and 2 Nexus 7 devices) and we change the average user presence time from 15 sec to 1 hour. To isolate the effect of presence time, once a device leaves our cluster an identical copy arrives and joins the cluster. Figure 4 shows the results of these experiments and compares femtcloud to the presence time Oblivious scheduler (PreOb). Figure 4(a)



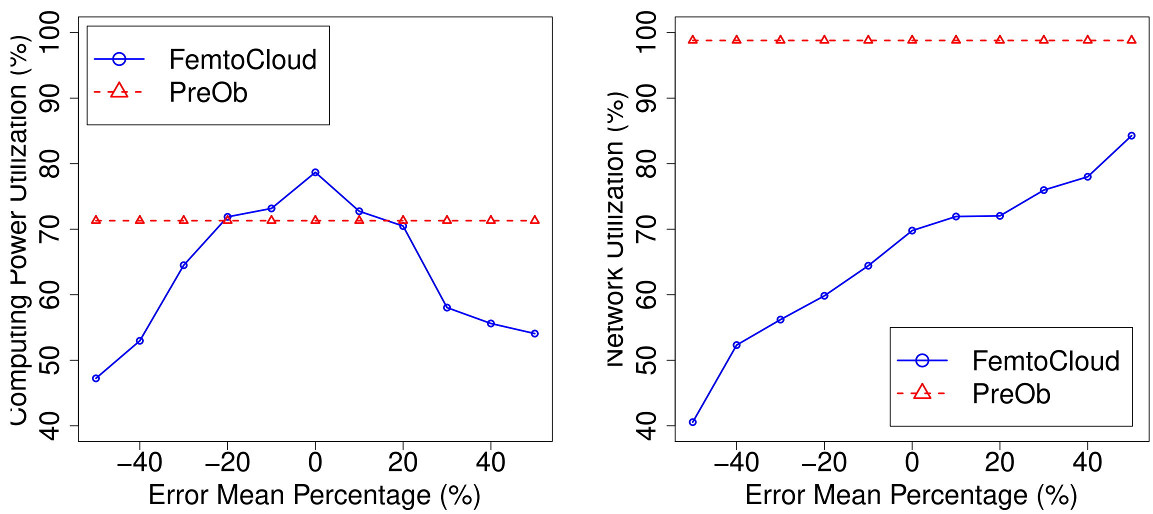
(a) Computational Resource Utilization

(b) Network Utilization

Figure 5: Task characteristics impact.

and Figure 4(c) shows that with the increase of the presence time, both algorithms utilizes the stability to gain more performance. femtocloud's awareness of the presence time enabled it to achieve higher performance than PreOb for low presence time values. Figure 4(b) shows that the femtocloud's increased performance comes with lower network utilization. The main reason is that without the knowledge of the presence time, PreOb assigns tasks to devices that may not be able to execute them and, thus, it may have to reassign them again to another device. This behavior keeps the network unnecessary busy. Figure 4(b) also shows that with the increase of presence time femtocloud becomes able to execute tasks that require high compute resource and low network usage. Therefore, the more stable the devices in the femtocloud the less it consumes from the network resources.

Task characteristics impact: To study the impact of changing the task characteristics, we conduct an experiment that has only single type of tasks (Chess). While maintaining the average computational requirements and average output size as constants, we vary the input size from 0.5 MBytes to 16MBytes. Figure 5 shows



(a) Computational Resource Utilization

(b) Network Utilization

Figure 6: Robustness to estimation errors.

the impact of increasing the task input size on the performance of femtocloud. It is clear that with the increase of the input size, the task characteristics moves from being CPU bounded, which enables increasing the compute resource utilization, to be fully Network bounded. Therefore the compute resource utilization decreases and the network utilization increases.

Robustness to estimation errors: To measure the impact of errors in estimating the presence time of the user on the system, we conduct an experiment in which we introduce an Gaussian error and changed the mean from -50% of the presence time to +50% of the presence time. Figure 6 shows that when the error mean is 0, femtocloud is able to achieve the highest utilization of the available devices. When the error is negative, we have a conservative estimate about the presence time which limits femtocloud usage of a device, which leads to decreasing the compute and network utilization. When the error is positive the computational utilization degrades because femtocloud fails to gather all the executed task results. Note that our early gathering heuristic is responsible for minimizing this effect. Figure 6(b) shows that the network

Table 5: Prototype performance measurements

Scenario	Oracle	femtocloud
Full presence	16.54 MFLOPS	14.23 MFLOPS
Emulated arrival/departure	10.31 MFLOPS	8.86 MFLOPS

utilization keeps increasing because femtocloud keeps assigning tasks more and more tasks while moving from a conservative estimate to a less conservative one. Further more, when the error becomes positive, the network utilization will further increase due to reassigning tasks after a device leaves without sending their results.

3.4.3 Femtocloud Prototype Evaluation

In this section, we discuss the results we gathered while using our prototype. In our experiment, we use three devices, a Galaxy S5 running Android 4.4.4 in addition to a Nexus 10[2013] and a Nexus 7[2013] tablets running Android 5.0.2. In this experiment, we compare the performance of femtocloud to an oracle which assumes accurate knowledge of all connectivity and execution time for every task on every device. Since this oracle is impossible, we gather measurements from all the devices and use after the fact analysis get the results.

In our experiment, we compare the achieved compute throughput by the oracle and femtocloud under two scenarios: (1) Full presence scenario, and (2) Emulated arrival/departure scenario. In the first scenario, we assume that the three devices existed during the whole period of experiment (1 hour). The main goal of this scenario is comparing the maximum achievable performance of femtocloud to the one achieved by the oracle. In the second scenario, we emulate average presence time of two minutes for each device. We emulated the arrival of new devices by returning the device to the cluster after average of one minute from its last departure. Table 5 summarizes these experiment results and shows that femtocloud achieved more than 85% of what the oracle achieved in both scenarios.

3.5 Summary

In this chapter, we have designed, implemented, and evaluated the FemtoCloud system that leverages the available compute capacity on a collection of co-located mobile devices to form an edge compute resource. We presented the design and architecture of the system. We identified the task scheduling problem as an important part of the design of such a system and developed an optimization framework that led us to scalable heuristic solution to the problem. Our evaluation demonstrated the potential for femtocloud clustering to provide a meaningful compute resource at the edge.

As mentioned previously, building the FemtoCloud system that forms a meaningful and efficient compute resource out of a single cluster of mobile devices with churn lays the foundation towards building a mobile-cluster based compute service provider. In the following chapters, we are building on this FemtoCloud system and extend it in multiple directions.

CHAPTER IV

WORKLOAD MANAGEMENT IN EDGE FEMTOCLOUDS

4.1 Introduction

In this chapter, we build on our previous work, presented in chapter 3, by addressing the full requirements of workload management in Femtoclouds and the system coverage beyond a single mobile device cluster. At a high level, these functions enable a Femtocloud to provide a service to *job originators* that is comparable to that provided by a centralized cloud service, namely the submission of jobs for completion in a timely and reliable manner[1]. Further, because the Femtocloud comprises mobile device *helpers*, the system must provide an interface for these devices to opt in and out. Under the covers, the system must manage a continually changing pool of helpers, assigning and moving computational work in response to job demands and device churn.

We begin with the observation that selective use of the cloud for control and management allows a Femtocloud to retain deployment advantages while significantly increasing the opportunity to provide a stable interface to both job initiators and willing helpers. In particular, a *controller* in the cloud can serve as the persistent and well-known contact point to receive jobs for processing, to receive helper requests to join, and to monitor helpers during job execution.

A cloud-based controller is a good starting point towards a stable service, but a

[1]Some cloud services provide dedicated servers. That is clearly not possible with mobile device clusters, and instead we focus on a comparable job processing service.

collection of additional workload management functions are needed to further overcome and mask the effects of device churn. These functions bear some similarities to those used in traditional computation services (e.g., admission control, job/task assignment), however they depart from tradition by making primary the assumption that compute resources are highly dynamic.

These functions are required to enable the system to receive compute jobs from *job originators* and enable mobile devices called *helpers* to process them in a reliable and scalable manner. The system should also be able to handle a large class of jobs including those that consist of multiple interdependent *tasks*, where the dependency can be modeled with a directed acyclic graph (DAG). Mobile devices should be able to opt in the system to act as helpers and share their resources at any point in time regardless of their location. The system has to efficiently handle churn of helpers since devices may leave at any point in time due to user mobility, resource limitations, lack of connectivity and/or energy constraints.

In this chapter we develop a modified system architecture that relies on the cloud to efficiently control and manage a Femtocloud. Within this architecture, we develop adaptive workload management mechanisms and algorithms to manage resources and effectively mask churn. These mechanisms include: (1) An efficient admission control mechanism designed to maintain system stability and avoid helper overload. (2) A task assignment algorithm that incorporates and adapts to critical path scheduling to suit the highly-dynamic and heterogeneous environment inherent in mobile device clusters. (3) A checkpointing mechanism to avoid losing the results of critical tasks after being completed due to helper churn. (4) A helper queue management algorithm that ensures fair resource allocation between the jobs that share the same helper.

We implement a prototype of our Femtocloud system on Android devices and utilize it to evaluate the overall system performance. We also use simulation to isolate and study the impact of each of our workload management mechanisms, and test the system at scale. Our prototype results demonstrate the efficiency of the Femtocloud workload management mechanisms specially in situations with potentially high churn. In particular, when the helper churn is relatively high, the Femtocloud workload management mechanisms workload management can reduce the average job completion time by up to 26% compared to the CPOP scheduling algorithm [61] which is used in traditional cloud computing systems. In larger scale experiments, our simulations showed that our admission control mechanism maintains the stability of the system regardless of the job arrival rate. In addition, using our checkpointing mechanism further reduces the average job completion time achieved by our task assignment mechanism by up to 31%.

The rest of this chapter is organized as follows. Section 4.2 describes the details of the Femtocloud system architecture. Section 4.3 develops the details of the workload management functions within the Femtocloud architecture. Section 4.4 show results that assess the efficiency of each of our workload management mechanism independently. Section 4.5 presents details of our prototype implementation and the performance evaluation of the whole system in a small-scale scenario. Section 4.6 addresses the challenges associated with large scale deployment. In particular, it discusses how to provide users with incentives to share their compute resources and surveys mechanisms to protect the Femtocloud architecture against malicious helpers and/or job originators. Finally, Section 4.7 summarizes our chapter.

4.2 System Architecture

In this section, we provide an overview of the Femtocloud system and its main architectural components depicted in Figure 7. It consists of three main components: a *Femtocloud controller* running on the cloud that manages the available compute resources and provides the initial interface to job originators and helpers; a set of

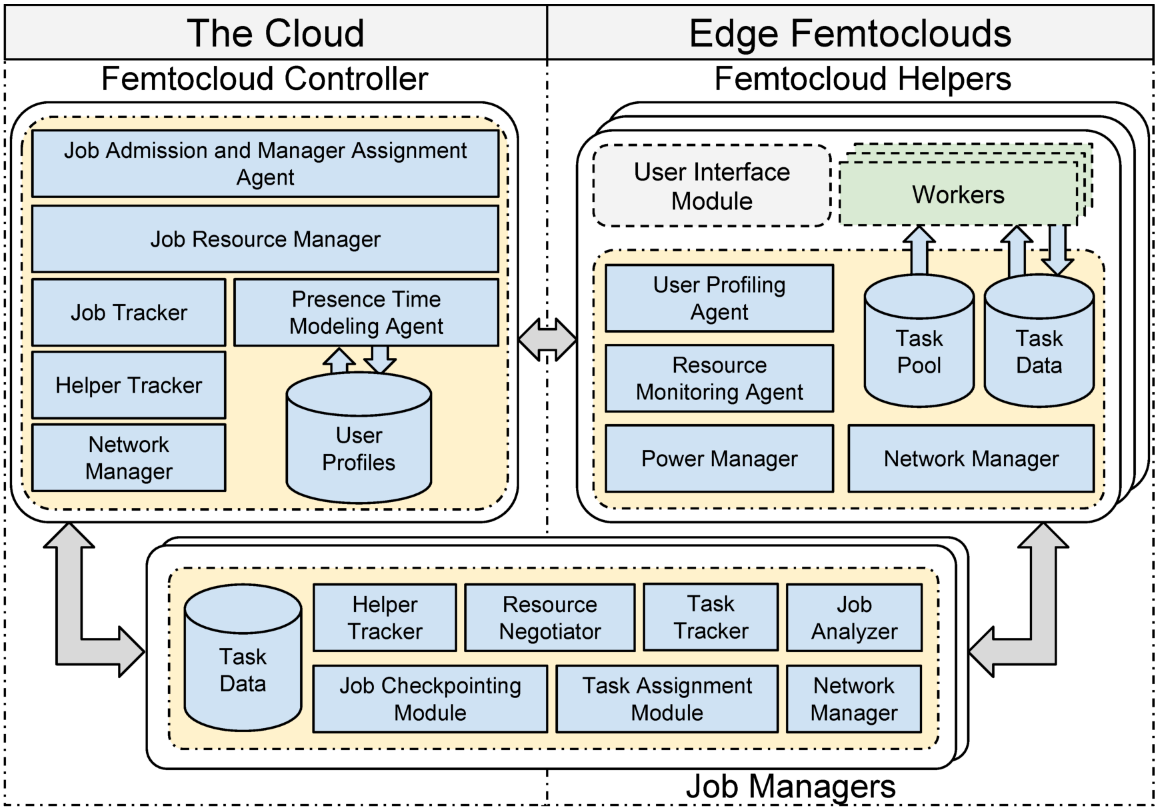


Figure 7: System architecture for edge Femtoclouds.

mobile devices running a *helper* client application and sharing portions of their compute resources; a set of *job managers* each of which is responsible for managing only a single job, taking over the interaction with the originator and the helpers for a certain job after the controller has accepted the job. Hence, job managers provide functional separation between the work to accept a job and the work to manage a job to completion. For scalability, they allow the work of run-time job management to be distributed. A job manager may be located in the cloud or, if appropriate, at a helper willing to take on more than just job computation. Furthermore, if a job manager can be located near its job originator and/or its helper set, rather than in the cloud, there may be performance advantages.

We assume that each job is represented by a directed acyclic graph (DAG) where each node in the graph is a *task* and a directed edge indicates a completion dependency. Each task node is labeled with an estimate of the computational requirement of the task; each edge is labeled with an estimate of the communication requirement to send the task results to downstream tasks that depend on them. This job model covers a wide range of applications, though clearly not all possible jobs. In particular, jobs whose task structure and requirements change at run-time do not fit this model.

We now provide the details of the three main architectural components (depicted in Figure 7).

4.2.1 Femtocloud Controller

The Femtocloud controller provides a registration and management interface for users who have a job to execute or helper resources to share. To maintain system reliability, the Femtocloud controller is deployed on a commercial cloud (e.g., Amazon EC2 or Windows Azure). Figure 7 illustrates the main components of the Femtocloud controller. To support helper management, the controller periodically collects information about helper status, user profiles, and resource availability through the **helper tracker** function. It uses the gathered information to model individual device churn using the **presence time modeling agent**. These models are later shared with the job managers to be used for task assignment and job management purposes. The controller also collects information about a helper's network connectivity and use it in deciding which helpers to use for which jobs. As a simple example, a job that requires a large amount data to be moved between between dependent tasks is ideally scheduled on helpers that are close to one another (in network terms).

In addition to helper monitoring, the controller is also responsible for job admission and resource allocation. When a new job arrives, the **job admission and manager assignment agent** first decides whether or not the job can be accepted and executed by the Femtocloud cluster using the algorithm presented in Section 4.3.1. If accepted, the controller spawns a job manager in the cloud to handle the recently accepted job. The controller may then migrate the job manager to the task originator or one of the helpers to enhance communication efficiency and increase responsiveness. It periodically communicates with the job manager to collect job progress updates through the **job tracker**. It also handles the job manager's resource allocation requests using the **job resource manager**.

4.2.2 Femtocloud Helpers

A Femtocloud helper is a client application responsible for executing assigned computation tasks. A user first needs to configure his/her personalized resource sharing policy and privacy requirements via the **user interface module**. These policies influence the behavior of the **user profiling agent**, by dictating what is allowed and not allowed with respect to monitoring user mobility and device usage patterns. User profiles are generated and shared with the Femtocloud controller based on user-defined privacy constraints. Additionally, the helper monitors the CPU and memory usage using the **resource monitoring agent**, and implements the **power manager** functionality which tracks the available battery level on the device. All this information is periodically shared with the controller and any job managers to which the helper has been assigned.

To support task assignment that is cognizant of network performance, each helper implements a **network manager** that monitors the available network interfaces to estimate the bandwidth and the round trip time (RTT) while communicating with other entities (controller, job originators, and other helpers).

Each helper has a pool of tasks assigned to it by job managers. It also has a set of worker modules each of which is in charge of a single shared CPU resource (e.g., core). Once a CPU resource becomes available, the worker in charge of that resource picks up a task from this pool using the algorithm described in Section 4.3.3 in order to provide fairness among different jobs. Once the task completes execution, the worker writes the task's results and state information to the task data storage.

4.2.3 Job Managers

A job manager is responsible for handling only one job and dealing with the job's originator. It starts as a service at the controller and can then be migrated to one of the helpers or the job originator to enhance communication efficiency and increase responsiveness while handling originator requests and/or managing helpers. Once started, a job manager analyzes the task dependency graph of its associated job to determine its critical portions and overall resource requirements using the **job analyzer**. Based on this information, the **resource negotiator** contacts the controller seeking resources as needed. We describe the details of the resource allocation and negotiation process in Section 4.3.2.2. Once a set of helpers are assigned to the job manager by the controller, the job manager periodically collects information about the available resources at these helpers through the **helper tracker**. It also collects information about the helper network connectivity using the **network manager**.

The job manager's main function is to quickly complete the associated job and return the results to its originator. To achieve this goal, the **task assignment module** uses the available information about tasks and helpers to assign tasks accordingly. We present the details of the task assignment mechanism in Section 4.3.2.3. It, further, tracks the progress of the assigned tasks using the **task tracker**. To mitigate the effect of churn and avoid re-executing tasks upon a departure of a helper, the **job checkpointing module** selectively backs up the results of a subset of the completed tasks on a selected set of helpers as well as the cloud using the algorithm described in Section 4.3.4.

4.3 Workload Management

A brief scenario would best explain the breakdown and interaction between each of the mechanisms described in subsections 4.3.1 through 4.3.4. We begin with a Femtocloud controller running in the cloud and actively managing a group of helpers sharing some compute resources and running tasks assigned to them by a set of active job managers. When a new job is created at a job originator, this originator contacts the Femtocloud controller inquiring whether or not it can help execute this new job. The controller initially relies on our admission control mechanism presented in subsection 4.3.1 to decide whether to accept or reject this job. If accepted, the controller instantiates a new job manager to assign job tasks to various helpers as described in subsection 4.3.2. This new job manager analyzes the task dependency graph of the job and identifies its critical sections, divides the job into stages to mitigate churn (i.e., impact of helper/resource departure), requests resources on-demand, and distributes tasks across available helpers accordingly. Since a single helper can be assigned to multiple job managers, we use the algorithms described in subsection 4.3.3 to achieve a fair compute resource distribution across jobs on a given helper. Finally, once a helper finishes executing a task, the job manager may decide to replicate the results of the task on multiple helpers and/or in the cloud to avoid the need for re-executing these tasks to further mitigate helper churn. This checkpointing mechanism is described in subsection 4.3.4.

All our mechanisms and algorithms rely on the structure of the job DAG and on estimates of job computation and data requirements. The set of parameters representing these requirements is shown in Table 6. Note that the DAG structure of jobs has been thoroughly studied and is widely used [51]. In addition, we only need estimates for computation and data parameters to guide our decision making. These estimates can be obtained using techniques such as those outlined in [17]. The effect of errors in these estimates is evaluated in Section 4.4.

Table 6: Job Parameters

Symbol	Description
c_T	Compute (processing) requirement of task T
o_T	The size of the output of task T
e_T	The size of the executable code coupled with the external data needed by task i
d_{Tk}	Determines whether task T requires the output of task k to start executing (1) or not (0)
f_T	Determines whether the helpers have finished executing task T (1) or not (0)

4.3.1 Job Admission Control

A Femtocloud strives to minimize the job completion time and enhance its users' quality of experience under the constraint of not overwhelming the helpers. Therefore, it is critical for the Femtocloud controller not to accept incoming jobs that will overwhelm the helpers and are beyond the system capacity. There are many options for the admission control policy, and our aim in this paper is not to explore them in detail. Instead, we use a simple job admission policy in which the controller accepts new jobs based on progress towards completion of the jobs already in the system. Specifically, if the total relative work remaining on existing jobs is above a threshold, the new job is rejected. An appropriate value for the threshold can be learned and adapted over time using measurements of completion time for jobs, average level of work parallelization for jobs, number of helpers in the system, and average helper utilization.

To apply our admission control policy, only the job progress information is periodically reported from active job managers to the controller. The current relative progress of a job is calculated at the job manager using the following equation:

$$\rho = \frac{\sum_T f_T c_T}{\sum_T c_T}$$

where ρ is the relative job progress, $\sum_T f_T c_T$ is the computational load of the fully executed tasks, and $\sum_T c_T$ is the job's computational load.

Although this method values progress on short and long jobs equally with respect to the admission decisions, the size of the job is implicitly taken into account since shorter jobs will progress faster than longer ones. For instance, if all the arriving jobs are relatively short, it is expected that they will progress faster and thus more jobs will be admitted over time. On the other hand, if all the jobs are relatively long, less number of jobs will be admitted over time due to the slow progress of the jobs.

4.3.2 Single Job Task Assignment

In this section, we develop a task assignment algorithm that takes a single job, represented by a DAG, and assigns its tasks to helpers such that the job completion time is minimized. The algorithm consists of two key mechanisms for (1) helper allocation and (2) task assignment. The helper allocation mechanism, presented in section 4.3.2.2, assigns the job a set of helpers that matches its requirements. The task assignment mechanism, presented in section 4.3.2.3, decides which task is to be executed by which helper. In the next section we deal with issues of fairness between jobs that share the same helpers.

Two conflicting considerations must be balanced in the assignment algorithm. The first is that helpers are ephemeral, thus suggesting a conservative approach to assigning tasks to a given helper, in case it leaves the system before completion[2]. The second is that there may be significant communication costs with transferring the output of one task to those downstream, thus suggesting that tasks with data dependencies should be scheduled on the same helper. On the other hand, the fact that different jobs may have different bottlenecks suggests that the task assignment algorithm should take the task requirements and its location in the DAG of the job into account while making any assignment decision. Finally, the churn in helpers suggests that overall the algorithm should schedule tasks in batches, retaining the

[2]We later discuss the use of checkpointing to help with this issue. Checkpointing has costs, however, so reducing the need for checkpointing is important.

ability to adapt as the job executes.

4.3.2.1 Critical Path and Stages

At a high level, tasks in the job dependency graph form execution paths, each of which consists of a set of tasks that has to be sequentially executed. The path with the highest total computation requirement is referred to as the *critical path*. As tasks complete, the remaining computation load in each path may change. Therefore, we use a notion of the *current critical path*, i.e., the path of tasks in the job dependency graph with the highest total remaining computation requirement.

As shown in Figure 8, we refer to the last task of the job that marks its completion as *the exit task*. Formally, the computational requirements of the current critical path to the exit task T_{exit}, ($\mathcal{C}_{cp}(T_{\text{exit}})$) can be calculated using the following formula:

$$\mathcal{C}_{cp}(T) = \begin{cases} c_T + \text{argmax}_k \left[d_{Tk} \mathcal{C}_{cp}(k) \right] & \text{if } f_T = 0 \\ 0 & \text{if } f_T \neq 1 \end{cases}$$

where $C_{cp}(T)$ is the compute requirements of the critical path leading to task T including the task itself, c_T is the compute requirements of task T, f_T determines whether task T has already been completely executed (1) or not (0), and d_{Tk} determines whether task T requires the output of task k to start running (1) or not (0).

A subset of the tasks in the current critical path has no unfulfilled incoming data dependencies and are ready to be executed once assigned to a helper. We refer to this set of tasks as the *ready section*, which we prioritize when we schedule tasks. Figure 8 shows an example that illustrates the difference between the critical path and the ready section for a job. Progress on the critical path will be important to scheduling on-critical-path tasks. In particular, we will keep track of the *relative critical path progress* from executing a ready section:

$$\mathcal{P} = \frac{\sum_{T \in S_{\text{ready}}} c_T}{\mathcal{C}_{cp}(T_{\text{final}})}$$

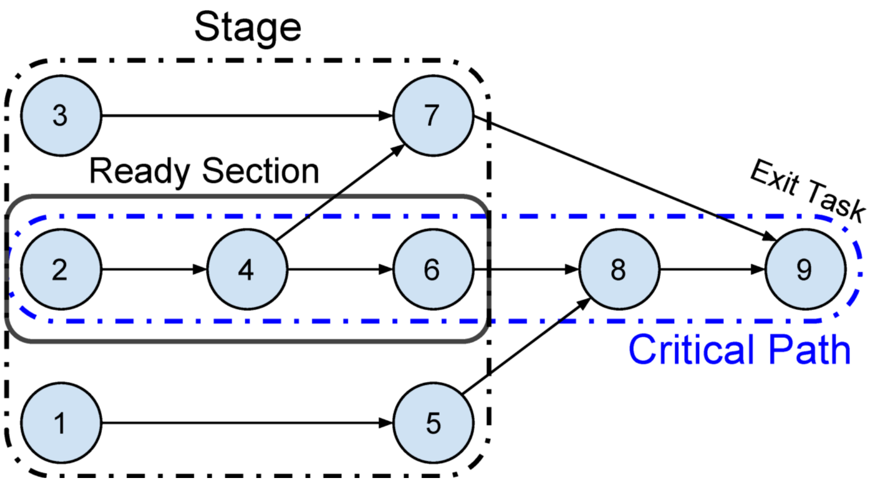


Figure 8: Critical path, ready section, and stage illustrative example. For simplicity, all tasks are assumed to have equal computational demand

where $\mathcal{P}$ is the relative critical path progress and S_{ready} is the ready section's set of tasks.

In the absence of helper/resource churn, as in most classical cloud environments, it is clear how to assign tasks to resources once the critical path is identified. The critical path tasks can be assigned first and then all the non-critical path tasks can be assigned to the resource that is going to finish them fastest [61]. The assignment decision, however, is not as straightforward in a Femtocloud due to potentially high churn. For instance, rushing into assigning and executing tasks that do not belong to the critical path before their results are actually needed may lead to result loss (if the helper leaves) and the need to re-execute.

Lemma 1 *For every non-critical path in the job dependency graph, the path's computational progress will not affect the job completion time if the relative progress of the path, at any point in time, is at least equal to the critical path's relative progress.*

Based on Lemma 1, we highlight that an efficient job assignment strategy should (1) strive to achieve high progress in the critical path, and (2) maintain the relative progress of any non-critical path to at least be equal to the relative progress of the critical path. In addition, we argue that the assignment strategy should also avoid achieving relatively high progress in any non-critical path to mitigate the risks associated with the churn of helpers.

To achieve these goals, we define a **stage** as the smallest set of tasks that include (1) the current critical path's ready section, and (2) all the tasks, belonging to non-critical paths, that are needed to keep the relative progress of these paths greater than or equal to the relative progress of the critical path. Figure 8 shows an example that illustrate our main concepts. The main objective of our task assignment mechanism thus can be reduced to minimizing the time needed to finish the execution of all the tasks that belong to the current stage.

4.3.2.2 Helper Allocation

The controller is responsible for managing the complete pool of helpers and for allocating them to job managers dynamically. A job manager requests helpers from the controller using one stage lookahead, to avoid delay when the next stage is ready to start. The job manager estimates its need for helper capacity in the next stage by estimating the expected stage-level parallelism using the following equation:

$$E_p = \left\lceil \frac{\sum_{T \in S_{\text{stage}}} c_T}{\sum_{T \in S_{\text{ready}}} c_T} \right\rceil$$

where E_p is the expected stage-level parallelism ($E_p \geq 1$), S_{stage} is the set of tasks in the target stage, S_{ready} is the set of tasks in the critical path ready section of that target stage, and c_T is the compute requirement of task T. In example illustrated in Figure 8, the expected stage-level parallelism equals 2.33 and the target number of helpers equals 3 (the ceiling of E_p) .

The job manager compares the target number of helpers for the next stage to the current set of helpers and decides to request/release helpers accordingly. If the manager decides to release helpers, it releases the ones with the least expected compute resource availability (the allocated compute capacity multiplied by the expected presence time). Otherwise, it sends a request to the controller with (1) the target number of additional helpers, (2) the target helper's compute resource availability of the ready section compute requirements ($\sum_{T \in S_{\text{ready}}} c_T$).

Once the controller receives a resource allocation request from a job manager, it checks the availability of helpers that satisfy the requested compute requirement based on their (1) expected presence time, (2) shared compute capacity, and (3) other-job commitments. Helpers that satisfy the compute requirement with the minimum average latency while communicating with the originator, job manager, and already assigned helpers are selected and assigned to the job. If the number of helpers that satisfy the requested compute requirement does not fulfill the job manager's request, the controller will assign all of available helpers, if any, to the job manager to enable it to start tasks as soon as possible. It will also maintain the job manager's request and assign additional helpers as they join the Femtocloud.

4.3.2.3 Risk-Controlled Task Assignment

Once a stage is ready to be started and its needed resources are allocated, the task assignment process begins. To assign the stage tasks to their helpers, we adopt a path-based assignment policy to minimize the overhead of moving data between helpers. We implement a path selection mechanism to pick paths from the current stage to be directly assigned to their helpers. This mechanism iterates on all the paths of unassigned tasks with fulfilled dependencies and selects the one with the highest total compute requirements to be assigned first. This mechanism is repeated until all the tasks in the stage are assigned to helpers.

When a path of tasks is ready to be assigned to a helper, we use Algorithm 1 to select the helpers to which the tasks should be assigned. In this algorithm, we first estimate the amount of time needed to finish all the tasks in the path on each helper based on (1) the amount of time needed to send each task coupled with its input data to the helper, and (2) the execution time of all the tasks based on the helper capacity.

Algorithm 1 Task Assignment

```
procedure AssignTasks({T}, {H})          ▷ T is of a task type. H is of a helper type
    selectedH ← null; lowestRisk ← +∞;
    for H in {H} do
        compTime[H] ← H.completionTime({T});
        churnProb[H] ← H.churnProbability(compTime[H]);
        if lowestRisk > churnProb[H] then
            selectedH ← H;
            lowestRisk ← churnProb[H]
        end if
    end for
    SortedH ← sorted({H}, churnProb)                        ▷ Risk based sorting
    for H in {SortedH} do
        if compTime[H] > compTime[selectedH] then
            continue;                       ▷ high risk, high compute time helper
        end if
        AddedRisk ← (churnProb[H] − churnProb[selectedH]) / churnProb[H];
        Gain ← (compTime[selectedH] − compTime[H]) / compTime[selectedH];
        if Gain > AddedRisk then
            selectedH ← H;
        end if
    end for
    return selectedH
end procedure
```

Let us use T_{ch} to denote the path completion time of the path on helper h. We also use $\mathcal{T}_{ph}$ to denote the total presence time of helper h measured from its arrival time. Based on the estimated completion time and the helper's churn model, we estimate our risk factor, called *churn probability* $P_h(\mathcal{T}_{ph} < (T_{ch} + S_h) \mid \mathcal{T}_{ph} > S_h)$, which is the probability that the helper will opt out of the system prior to completing the tasks. Note that S_h denotes the helper's time in the system up until this moment. The churn probability is computed using a model of the distribution of the time a helper spends in the system, also called *presence time*. This distribution can be learned using the User Profiling functions in the helper client application and/or learned on a system-wide basis by the controller.

Once the churn probability and the completion time are calculated for all the helpers, we sort the helpers according to their churn probability and the helper with the lowest churn probability is selected. Then, we iterate on the sorted list of helpers using a risk-controlled mechanism, commonly used in economics [52], to compare the gain of switching to this helper as the reduction ratio in the task completion time to the relative addition in the risk. In particular we use the following equations to calculate the gain and the risk:

$$G = \frac{T_{ch^*} - T_{ch}}{T_{ch}}$$
$$R = \frac{F(h, T_{ch}) - F(h^*, T_{ch^*})}{F(h, T_{ch})}$$

where G is the relative gain of using using h over the selected helper h^*, $F(h, T_{ch})$ is a function that calculate the churn probability of h^{th} helper ($F(h, T_{ch}) = P_h(\mathcal{T}_{ph} < [T_{ch} + S_h] \mid \mathcal{T}_{ph} > S_h)$) and R is the relative added risk introduced by using helper h over the selected one. If the gain exceeds the risk, it switches to the helper with the higher gain.

4.3.3 Multi-Job Helper Queue Management

Since the same helper can be assigned to multiple job managers and can execute tasks that belong to different jobs, determining the appropriate order in which the helper executes these tasks is important. A helper should be (1) predictable, allowing the job managers to make correct decisions, and (2) optimized to enhance the performance of the jobs that it contributes to.

4.3.3.1 Fair queuing based task pick up

To enhance predictability, we implement a fair queuing based task pick up mechanism, described in Algorithm 2. Each helper maintains multiple execution queues, each of which is associated with only one job. Each of these queues is associated with a credit counter used to insure fairness. The process starts when a helper becomes associated with a new job manager. In this case, the helper creates a new queue for the tasks that belong to the new job and assigns it zero credit. When a worker thread becomes

Algorithm 2 Helper Queue Management

```
procedure executeTask({Q})                                ▷ Q is a task queue type.
    adjustCredit({Q});
    selectedQ ← selectQueue({Q});
    execute(selectedQ.popHead());
    selectedQ.credit ← selectedQ.credit - ExecTime;
end procedure
procedure selectQueue({Q})                                ▷ Q is a task queue type.
    sortedQ ← sorted({Q});                                ▷ descending sorting on credit.
    earliestDeadline ← +∞; timeBuffer ← +∞;
    selectedQ ← null;
    for Q in sortedQ do
        if not Q.isEmpty() then
            continue;
        end if
        if Q.head.startingDeadline() < earliestDeadline and Q.head.exTime() < timeBuffer then
            selectedQ ← Q;
            earliestDeadline ← Q.head.startingDeadline();
            timeBuffer ← min(timeBuffer - Q.head.exTime(), earliestDeadline - now);
        end if
    end for
    return selectedQ
end procedure
procedure adjustCredit({Q})                               ▷ Q is a task queue type.
    maxCredit ← −∞ ;
    for Q in {Q} do
        if not Q.isEmpty() then
            maxCredit ← max(maxFreq, Q.credit);
        end if
    end for
    for Q in {Q} do
        Q.credit ← min(0, Q.credit - maxCredit);
    end for
end procedure
```

available at the helper, it invokes the "executeTask" function to pick up a task from these queues and executes it. To insure fairness, this function will pick up the first task in the queue that has the highest credit, executes it, decreases the credit of the queue by the amount of time taken to finish the task. To avoid credit drifts, it adjusts the queue credits maintaining the same relative difference with every pick up decision.

4.3.3.2 Deadline-based Optimization

As we described in Section 4.3.2.1, tasks differ in their urgency level depending on whether they belong to the critical path or not. Even within the same stage different paths may significantly differ in terms of their task compute requirements and their computation time on the helper to which they are assigned. Therefore, their task execution urgency may significantly vary. For instance less urgent tasks on a high capacity helper may tolerate delays without affecting their stage completion time. Such delay tolerance can be utilized to execute more urgent tasks and enhance the overall system performance. To utilize this fact, we rely on the job managers to set a starting deadline for each task while assigning it and extend the fair queuing based task pick up mechanism to take these deadlines into account while picking up tasks for execution.

To assign a starting deadline for a task, the job manager implements a two phase mechanism. First, while assigning the first path of a stage (the critical path's ready section) to a helper, we estimate the target stage completion time which is equal to the estimated path completion time. The second phase is activated while assigning all the remaining paths. In this phase, while assigning a path, tasks are assigned starting with deadlines derived from the target stage completion time and the helper's shared capacity.

At the helper, we implement an early pick up mechanism, as shown in Algorithm 2, where urgent tasks from queues with low credit can be executed before tasks from ones with higher credit **if and only if** they will not interfere with their deadline requirements.

4.3.4 Task Checkpointing

Section 4.3.2 presents two key approaches to mitigate the impact of helper churn prior to fully executing tasks. First, using the concept of execution stages avoids executing tasks and getting their results too early compared to when they are needed. Second, the risk-controlled task assignment minimizes the risk of helper departure prior completing the assigned tasks. Once the task results become available, however, it is essential to preserve them till they are used in order to further mitigate the effect of churn. Therefore, we implement a checkpointing mechanism with which the job manager may replicate the results of a selected set of finished tasks on a set of helpers and/or in the cloud. The main objective of this replication is to avoid losing the results of finished tasks.

Our checkpointing mechanism runs periodically (every 15 seconds in our implementation) and determines for a finished task whether new replicas need to be added, the current state needs to be kept as is, or the results need to be deleted since they are no longer needed. To describe our mechanism, let's use r_{Th} to indicate whether the results of task T exist on helper h (1) or not (0).

The process starts with estimating the probability of losing the results of each of the completed tasks in a specific period of time X using the following equation:

$$P(\text{loss-time}(T) < X) = \prod_{h=0}^{m} [1 - r_{Th} (1 - F(h, X))]$$

where loss-time(T) is the time needed to lose all the replicas of task T, and $F(h, X)$ is a function that calculate the churn probability of h^{th} helper ($F(h, X) = P_h(\mathcal{T}_{ph} < [X+S_h] | \mathcal{T}_{ph} > S_h)$) during a period X given the helper's prior stay of S_h. Note that if a replica of the results of task T exist on the h^{th} helper ($r_{Th} = 1$), $[1 - r_{Th} (1 - F(h, X))] = F(h, X)$. However, if the helper does not have a replica of the task results ($r_{Th} = 0$), $[1 - r_{Th} (1 - F(h, X))] = 1$. We set X to be equal to twice the time needed to re-execute the checkpointing mechanism ($X = 30$ seconds in our implementation).

A task is considered *well-maintained* if its loss-time probability is less than a *reliability threshold ($\mathcal{K}$)*, which is set based on the environment and target level of reliability. To decide if tasks have to be replicated to reach a well-maintained status, we iterate over the not-well-maintained tasks in order of their loss-time probability and calculate the amount of computation (E_T) needed to reconstruct their results from the set of well-maintained tasks. We compare E_T to a linear function of the result size of the task (a_T) and decide accordingly whether the task needs to be replicated or not. Note that the coefficients of this function are environment dependent and based on the available bandwidth between helpers and their average shared capacities.

Once all tasks that require replications are marked, we re-iterate on them in order to determine the ones no-longer needed, relative to the current set of well-maintained tasks. We then issue replication requests for the ones that require additional replication followed by a delete request for the results of the tasks that are no longer needed to free up storage at the helpers. To replicate a task, we iterate over helpers in descending order of their churn probability and assign a replica to a helper if and only if it has available storage. This process is repeated until the loss-time probability becomes lower than the reliability threshold $\mathcal{K}$.

4.4 Mechanism Evaluation

In this section, we evaluate the performance of Femtocloud and assess the efficiency of its workload management mechanisms. We use simulations to isolate the true impact of each mechanism individually. We simulate different scenarios and environments, analyze the impact of using various management mechanisms, and study the effect of different parameters on the performance of the Femtocloud system. We start by describing our experimental setup (Section 4.4.1) followed by representative simulation results (Section 4.4.2).

4.4.1 Experimental Setup

We start by describing the experimental job model followed by the summary of the characteristics of the set of mobile devices used in our experiments. We then summarize our metrics and parameters followed by presenting our baselines.

4.4.1.1 Job models

In our experiments, we use a set of synthesized jobs to evaluate the performance of our workload management mechanisms. To construct the task dependency graphs for these jobs, we use a set of models that represent a large variety of application and programming paradigms. We focus on the following four representative job models:

- **Pipeline Job Model:** All the tasks in the job form a single path and must be sequentially executed. This model represents a wide range of single threaded jobs and/or applications.

- **Parallel Path Model:** Tasks form p parallel paths with very limited inter-path dependencies (set at a 0.1 probability). This job model represents multi-threaded applications with minimum inter-thread dependency and synchronization.

- **General Parallel Path Model:** Tasks form p parallel paths with more significant inter-path dependencies (set at 0.4 probability). This job model reflects general multi-threaded applications.

- **Pyramid Job Model:** Tasks form a tree structure where the task dependency direction goes from leaf-nodes towards the root. This model encompasses a wide range of map-reduce jobs.

To construct a job that follows one of these models, we first generate a fixed number of tasks. Each of the generated tasks falls in one of four categories: (1) lightweight tasks, (2) medium tasks, (3) compute intensive tasks, and (4) data generating tasks. Based on the selected category, we pick the computational requirements of the task and the output size from a normal distribution with the mean values listed in Table 7. Once all the tasks are built, we set the dependency edges between them using a pseudo-random process based on the target job model

Table 7: Experimental tasks' characteristics.

Task Type	Computation	Output
Lightweight tasks	10 MFLOPs	0.2 MBytes
Medium tasks	30 MFLOPs	2 MBytes
Compute Intensive tasks	100 MFLOPs	0.5 MBytes
Data Generating tasks	20 MFLOPS	20 MBytes

Table 8: Experimental helper's characteristics.

Devices	Computation Capacity
Galaxy S5	3.3 MFLOPS
Nexus 7 [2012]	7.1 MFLOPS
Nexus 7 [2013]	8.5 MFLOPS
Nexus 10 [2013]	10.7 MFLOPS

4.4.1.2 Device Characteristics

To evaluate the performance of Femtocloud under realistic helper characteristics scenarios, we identify the available compute capacity in a variety of mobile devices. We use matrix multiplication operations to emulate the compute capacity of a set of mobile and handheld devices. To achieve this, we measure the time needed to finish a load of 200 MFLOPs in a background thread using Galaxy S5, Nexus 7 and Nexus 10 devices and use the measured time to calculate the device compute capacity in MFLOPS. For each device, we repeat this process and take the average of 20 runs to measure the average compute capacity of the device. Table 8 summarizes the capacities we establish for the mobile devices we test.

4.4.1.3 Metrics and parameters

We are interested in the following performance metrics:

- **Job Completion Time:** This is the average amount of time needed to completely execute a job.
- **Job Admission Ratio:** This is the ratio of the number of admitted jobs to the total number of incoming job requests.

To assess the impact of each of our workload management mechanisms, we measure the performance of the system running our task assignment algorithm while turning other mechanisms (e.g., admission control, task checkpointing, early task pick up) ON and OFF.

4.4.1.4 Baselines

We compare our task assignment algorithm with the following baseline techniques:

1. **Critical path in a processor (CPOP)[61, 46]**: The CPOP scheduler aims to assign the critical path to the node that will execute it faster. To assign all the remaining tasks, it works in two phases. In the first phase, it assigns priorities to the tasks based on the amount of computations leading to them and the amount of computations after them. In the second phase, it assigns the highest priority task first to the node that will finish it faster.

2. **Per-task risk-controlled assignment (PTR)**: The PTR scheduler takes the scheduling decision for every task independently. Once a task is ready to execute (all its dependencies are fulfilled), it applies the same risk-controlled assignment mechanism as Femtocloud to select its executing helper. If more than one task is ready to execute, it assigns the one with the largest compute resource requirements first. This derived from the risk-adjusted return economic principle and is currently used by systems like COSMOS[53]

4.4.2 Results

In this set of experiments, we organize the helpers and the job originators in four groups each of which represents an enterprise network. The bandwidth and latency between two nodes in our experiment are modeled using a Normal Distribution where the standard divination is set to 20% the mean. The average bandwidth and latency between two nodes in the same group is 30 Mbps and 25 msec, respectively. However, the average bandwidth and latency between two nodes in different groups are 10 Mbps and 100 msec, respectively. We use a Poisson arrival process to model the arrival of new helpers. The helper arrival rate is set to 5 helpers/min and arriving helpers are randomly assigned to one of the groups. In addition, the helper presence is modeled as Normal(5 min, 1 min).

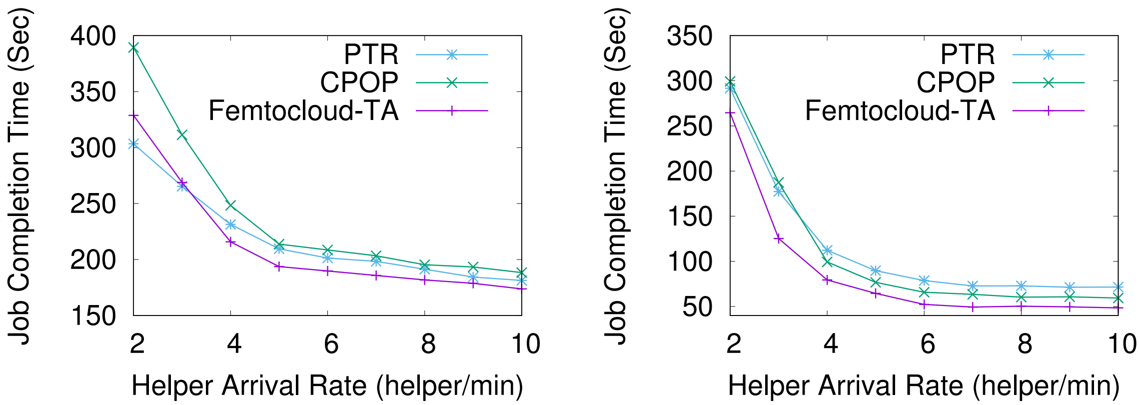
In this section, we only present the results from using the **General Parallel Path Model** and the **Pipeline Job Model**. We select these two models since (1) they cover a wide range of applications and real-jobs, and (2) they are considered the two extremes of our job model spectrum and they reveal all the interesting insights[3]. We use a Poisson arrival process to model the arrival of new jobs, where the job arrival rate is set to 5 Jobs/min. In addition, each job has 30 tasks that form its dependency graph. For the jobs that follow the **General Parallel Path Model**, we set the number of parallel paths to be equal to 5.

To assess the efficiency of each of our mechanisms independently, we start by disabling all our mechanisms and compare the performance of our task assignment mechanism to the base-line assignment mechanisms in Section 4.4.2.1. We then analyze the performance of our admission control mechanisms under different configurations in Section 4.4.2.2. Section 4.4.2.3 shows how using our task checkpointing mechanism helps in high churn situations. Section 4.4.2.4 analyzes the sensitivity of our workload management mechanisms to estimation errors. Each experiment represents the average of 10 runs.

4.4.2.1 Risk Controlled Assignment Performance

In this section, we study the impact of using our Femtocloud task assignment mechanism and compare its performance with the two baseline task assignment mechanisms

[3]We shed the light on the results from using the four job models in Section 4.5.

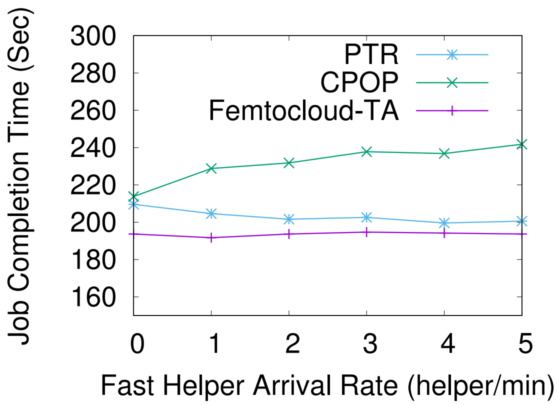


(a) Pipeline Jobs (b) General Parallel Jobs

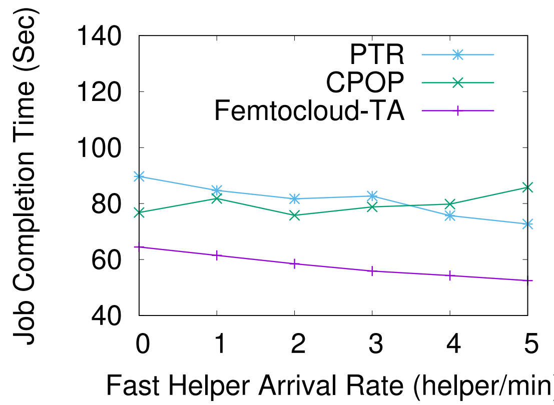
Figure 9: Impact of changing the helper's arrival rate on the task assignment performance.

(CPOP and PTR). To ensure fairness, we allocate all the helpers that join our system to every job manager while running each of the task assignment mechanisms. We focus in this section on the impact of (1) changing the helper arrival rate and (2) presence time heterogeneity. To avoid repetition, we will study the impact of the changing the average helper presence time in Section 4.5 using our implemented prototype.

Impact of changing helper arrival rate: Figure 9 shows the impact of the helpers arrival rate on the job completion time for the **Pipeline Job Model** and the **General Parallel Path Model**. The figure shows that when the helper arrival rate is low, resource sharing and competition between jobs increases and thus the average job completion time increases as well. Figure 9(a) shows that, in case of the pipeline job model, our Femtocloud task assignment mechanism (Femtocloud-TA) outperforms CPOP due to its ability to take the helper's presence time model into account while assigning tasks to them. It also outperforms PTR due to Femtocloud-TA's ability to assign multiple jobs back to back. However, when the helper arrival rate is low, making the number of helpers present at any point in time low compared to the number of jobs, assigning a full path of tasks increases the probability of



(a) Pipeline Jobs



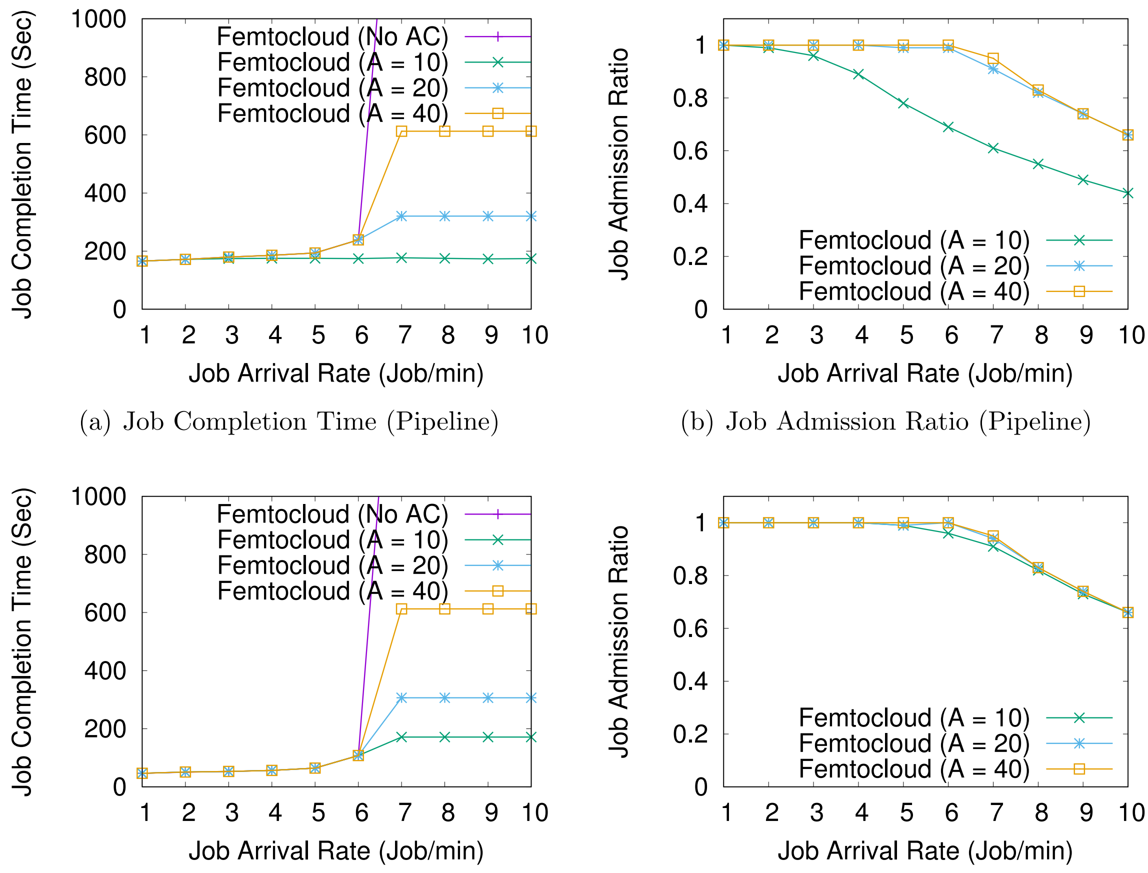
(b) General Parallel Jobs

Figure 10: Impact of helper's presence time heterogeneity on the task assignment performance.

losing the whole path and re-executing it which leads to wasting significant compute resources.

Impact of helper presence time heterogeneity: To understand the impact of helper presence time heterogeneity, we add a new set of helpers to our helper set. These new helpers (Fast Helpers) have the capacity of 10.7 MFLOPS and their presence times are modeled as Normal(30 Sec, 10 Sec). We use a Poisson process to model the arrival of these helpers. Figure 10 shows the impact of changing the arrival rate of the fast helpers.

Figure 10(a) shows that, in case of the pipeline job model, Femto-cloud-TA does not utilize the availability of the fast helpers due to the high risk of losing the job's computations due to the fast helper's churn. CPOP, however, did not take these helper's churn probability into account and assigned tasks to them. Such decision led to wastage of computation resources leading to an increase in the average job completion time. PTR, however, was able to utilize the additional capacity introduced by these helpers due to (1) its fine-grained per-task assignment mechanism, and (2) its ability to take the risk probability into account while assigning tasks to their executing helpers. Figure 10(b), however, demonstrates that the increased complexity of a job



(a) Job Completion Time (Pipeline)

(b) Job Admission Ratio (Pipeline)

(c) Job Completion Time (G-Parallel)

(d) Job Admission Ratio (G-Parallel)

Figure 11: Impact of job arrival rate on the performance of Femtocloud.

can result in its division into multiple stages allowing Femtocloud-TA to utilize the fast helpers to decrease average job completion time.

4.4.2.2 Admission Control Performance

We next study the impact of using our Femtocloud admission control mechanism. We compare the performance of using Femtocloud task assignment mechanism with and without using our admission control mechanism under different loads. In the following experiments, we disable (1) the checkpointing mechanism, and (2) the early task pick up mechanism.

Figure 11 shows the impact of changing the job arrival rate on the performance

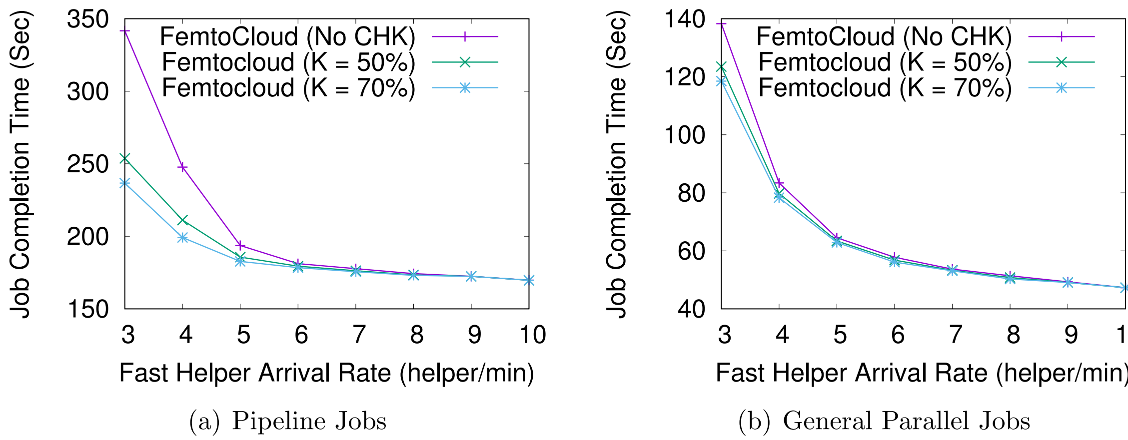


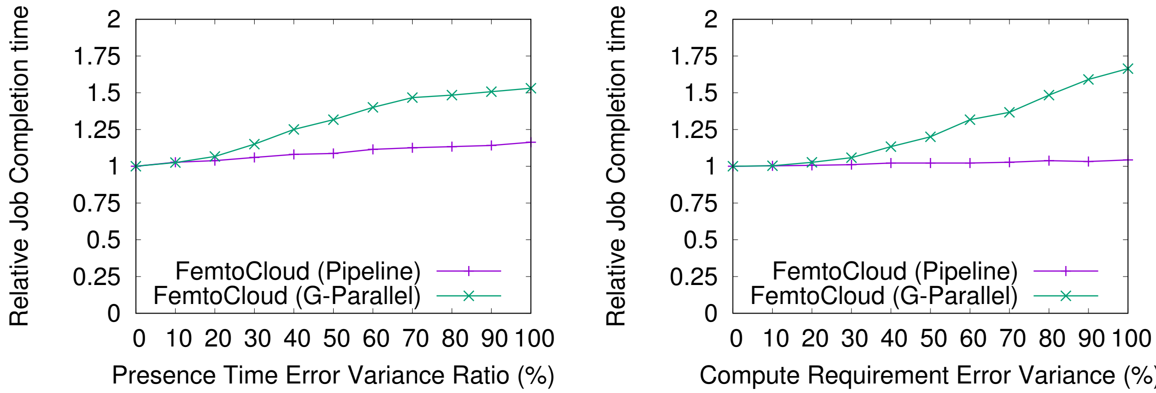
Figure 12: Impact of changing the helper's presence time on the checkpointing performance.

of Femtocloud with and without using admission control. While using our admission control mechanism, we experiment with the total relative work remaining maximum threshold ($\mathcal{A}_{\max}$) values of 10, 20, and 40. An increase in the job arrival rate causes the average job completion time to increase. Without admission control, we observe that the job completion time increases drastically once the number of jobs in the system exceeds its capacity. With admission control, job completion time can be limited to an acceptable value. It is important to carefully select the value of $\mathcal{A}_{\max}$. A low value of $\mathcal{A}_{\max}$ decreases the efficiency of the system and leads to the rejection of jobs that the helpers are capable of executing as shown in Figure 11(b) ($\mathcal{A}_{\max} = 10$).

4.4.2.3 Task Checkpointing Performance

We compare the performance of task assignment in Femtocloud with and without using task checkpointing while disabling all the other mechanisms.

Figure 12 shows the impact of changing the presence time of helpers on the performance of Femtocloud with and without using our checkpointing mechanism. We compare various reliability thresholds, ($\mathcal{K}$), when checkpointing is used. The figure shows that when the average helper presence time is low, there is increased need for



(a) Presence Time Errors

(b) Task Requirement Errors

Figure 13: Sensitivity to estimation errors (presence time model parameters and task compute requirements).

task checkpointing to avoid losing finished tasks, re-executing them, and increasing the overall job completion time. However, when the average presence time is large the probability of losing task results decreases and thus the checkpointing mechanism is not as important. Note that the checkpointing mechanism is more critical in case of the pipeline job model since the assigned path length increases.

4.4.2.4 Sensitivity Analysis

We analyze the sensitivity of our mechanisms to estimation errors. In this set of experiments, we enable all our mechanisms. We set the value of the reliability threshold ($\mathcal{K}$) for our checkpointing mechanism to be 70%. We analyze Femtocloud's sensitivity to estimation errors of (1) presence time model parameters and (2) task compute requirements. In both cases, our errors follow a Normal distribution with a mean of 0. We set the variance of the error distribution to be a percentage of the correct value and we vary this percentage from 0% to 100%. To show the error sensitivity, we report the ratio between the average job completion time with and without errors

Figure 13(a) shows, for both the pipeline job model and the general parallel job model, the job completion time increases with the increase of the presence time error variance. We notice that the pipeline job model is relatively less sensitive to errors in estimating the presence time model parameters because the checkpointing mechanism maintains copies of the intermediate task results leading to efficient recovery from the churn when it occurs. The figure also shows that the general parallel job model can sustain up to 20% error variance with approximately 10% increase in the average job completion time.

Figure 13(b) shows the impact of changing the variance of the compute requirement estimation error on the performance of Femtocloud. The figure demonstrates that the pipeline job model is insensitive to this type of error because the helper's fair queue management mechanism prevents a job task estimation error from influencing the performance of other jobs. The general parallel model, however, can accept up to 30% variance of the compute requirement estimation error without a significant increase in the job completion time.

4.5 Prototype Implementation and Evaluation

4.5.1 System Implementation

In this section, we present the implementation details of our Femtocloud prototype. We implement the Femtocloud controller logic as a python script that carries the responsibilities described in Section 4.2.1. We run this script on a local Linux machine. To emulate running on the cloud, we enforce an additional communication latency between the controller and the helpers of 168 ms, which is our measured average latency between a mobile device in Georgia Tech's enterprise network and Amazon Web Services in Europe (Frankfurt and Ireland).

We implement the Femtocloud helper as an Android application that allows users to enter their resource sharing policies. Based on these policies, the helper connects to the controller and shares the user profile accordingly. Upon joining the Femtocloud, the helper service estimates the mobile device capabilities and shares them with the controller. Additionally, while being used by a job manager, it shares with it the estimated fair share of resources that the job may receive from the helper. To ensure fair resource sharing between different jobs, it implements our mechanisms described in Section 4.3.3. It also carries the responsibility of estimating some contextual information as described in Section 4.2.2.

Our job manager is implemented as an Android service that can be assigned to the helper with the estimated longest presence time, or the job originator. The job manager's main responsibilities are described in Section 4.2.3. The state of the job manager is periodically replicated at the controller to enable recovery in case the device running the job manager functions opts out of the system.

Job originators are Android applications designed to generate jobs each of which consists of a set of tasks organized in directed-acyclic dependency graph. We emulate real tasks in the job by synthesizing a matrix multiplication task with the same input size, output size, and compute requirements of the target tasks. When a job manager assigns a task to a helper, the code is directly sent to the helper from the originator and is executed by the helper using the Java Reflection API.

In our prototype, we run the job originators inside Android x86 virtual machines. We configure the job originators to be willing to carry the job management responsibility. Therefore, once the controller accepts a job it starts the job manager service at the originator's VM.

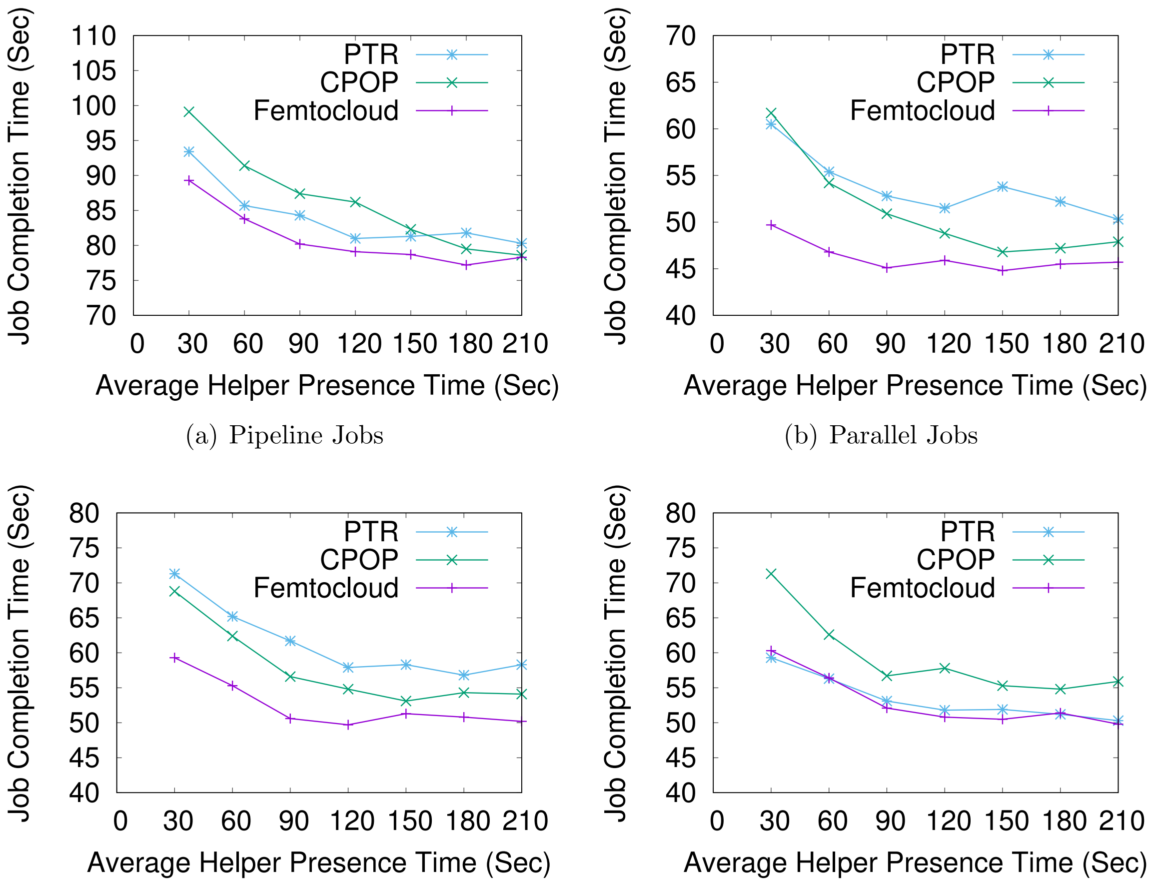
4.5.2 Results

In this section, we present the results acquired using our prototype implementation. Due to the limitations imposed by the scale of our experiments, we modified our checkpointing mechanism such that a task is considered well-maintained if its results are available in at least two helpers. We also set the number of full jobs allowed by Femtocloud to be relatively high such that all the incoming jobs are admitted by the controller. Therefore, we only present the job completion time results in Figure 8.

Our helper set consists of 6 devices, a Galaxy S5, a Nexus 10[2013], 2 Nexus 7 [2013], and 2 Nexus 7 [2012]. All these helpers are connected to the same enterprise network through WiFi. We use the Normal distribution to model the helper's presence time. In our experiments, we change the mean of the helper presence time distribution from 30 to 210 seconds. Once a helper leaves, it returns after an OFF period that follows a Normal distribution with mean equals 25% of the presence time mean maintaining the helper's duty cycle to be 75% on average.

We use a Poisson arrival process to model the arrival of new jobs. The job arrival rate is set to be equal to 3 jobs per minute. Each of the generated jobs consists of 15 tasks. We set the number of parallel paths in the jobs that follow the **General Parallel Path Model** to be equal to 3. In our results, we show the average of 5 runs.

Figure 14 shows the impact of changing the average presence time of the helper on the job completion time of different types of jobs. It is clear from the plots that with the increase of the helper presence time the job completion time decreases for all the scheduling mechanisms. Also all job models, Femtocloud outperforms the CPOP task assignment mechanism due to its ability to control the risk associated with assigning tasks to helpers that may leave before completing them. The relative advantage of Femtocloud over CPOP decreases with the increase of the helper presence time and decrease of churn probability. The figure also reveals that Femtocloud outperforms the PTR assignment mechanism under all job models except the pyramid model. For both the parallel and the general parallel job model, Femtocloud outperforms the PTR assignment mechanism due to its ability to identify the job's bottleneck (critical path) and prioritize it while assigning tasks to their executing nodes. For the pipeline job model, however, Femtocloud slightly outperforms PTR due to its ability to send multiple tasks back to back to the helper instead of waiting for each task to finish in



(a) Pipeline Jobs

(b) Parallel Jobs

(c) General Parallel Jobs

(d) Pyramid Jobs

Figure 14: Impact of helper's presence time on the performance.

order to assign the next one. In case of the pyramid job model, the Femtocloud loses its back-to-back assignment edge over PTR since all the paths at each every stage will consist of only one tasks. This will lead to Femtocloud schedule to operate on a per-task basis.

4.6 Discussion

In this section, we discuss two key questions associated the Femtocloud system: (1) How can the system provide users with incentives to share their mobile compute resources? (2) How can the system overcome the security challenges that will be introduced by malicious users (helpers/job originators). Although properly addressing those two questions are out of scope of our current work, they require some discussion.

4.6.1 User Incentives

We have witnessed increasing numbers in those willing to share their compute resources. For instance, BOINC projects demonstrate the willingness of millions of users to share the compute resources of their personal computers and mobile devices in support of scientific applications [1]. To achieve similar success, the Femtocloud system has to provide users with proper incentives to share their compute resources not only for scientific applications but also for a wide-range of edge computing applications.

To identify effective incentive mechanisms and assess users willingness to share their mobile compute resources while employing each of these mechanisms, we conducted a pilot user study. We surveyed approximately 50 students taking networking courses at Georgia Tech, at the undergraduate and graduate levels. Our survey consisted of two main sections. In the first section, we asked the students about the generic factors that might influence their decision when it comes to sharing their mobile compute resources (e.g., battery life, device type). These questions were intended to ensure the students' awareness to these factors prior to responding to the sharing-decision related questions. The latter section of the survey asked about their willingness to share in four specific scenarios: (1) in support of a for-profit company, (2) in support of gaming, (3) in support of science, and (4) in support of finding a lost child. In addition to the simple yes/no responses, students were asked to write additional comments, if needed.

Our pilot study reveals that individuals rationalize sharing their computational resources differently depending on who is utilizing the resources and why. All the students who were willing to share their resources with a for-profit business noted that they must receive some compensation (e.g., money). For the other three scenarios, the cause and the trustworthiness of the computation borrower were the driving factors behind their decision. The science scenario and the lost child scenario appealed to 76% and 83% of the demographic we surveyed, respectively. The gaming scenario, however, appealed to only 29% of the users surveyed.

The outcome of our pilot study suggests that people are willing to share their mobile compute resource if (1) they are getting compensated by the computation borrower, or (2) the cause of the computation is significant (e.g., common good, emergencies). Accordingly, a mix of these incentive mechanisms can be developed and integrated with our Femtocloud system to insure its adoption and future success.

4.6.2 Security and Privacy

In the Femtocloud system, ensuring the security and privacy of our helpers is critical for them to share their resources. Generally, helpers need to guarantee protection against malicious originators who may try to infringe on their privacy or compromise their security. Fortunately, using sandboxing allows the Femtocloud helper client service to control data access privileges and thus ensures helper data privacy [8, 28].

A Job originator must be protected against malicious helpers that may try to access the job's private data or provide incorrect results without executing the job. First, to protect against job-data leakage while running on untrusted resources, task execution over encrypted data was proposed [62, 24, 27, 42]. In these mechanisms, the originator encrypts the job's private data prior to submitting the job to the Femtocloud system. Upon the completion of the job, the Femtocloud system will return the results that only the job originator will be able to decrypt and understand. Second, to enable originators to verify the correctness of the results and and the work done by the helpers, cryptographically verifiable approaches can be used [26, 33]. These mechanisms enable the job originator to ensure the correctness of the results and verify that the tasks have been fully executed by the helpers. In the absence of these approaches, task replication over independent helpers can be used to detect inconsistencies and protect against malicious entities.

4.7 Summary

In this chapter, we presented an enhanced architecture for the Femtocloud system in which we rely on mobile device clusters at the edge to provide the compute resources while moving the cluster control and management functionalities to the cloud. Within this new architecture, we developed a set of adaptive workload management mechanisms and algorithms that make the Femtocloud system efficient for real computation workloads, and reliable and scalable in the presence of device heterogeneity and churn. We implemented a small scale prototype of our system and use it to demonstrate the feasibility and efficiency of the system. We used simulations to further understand the gains imposed by each of our workload management techniques in larger scale systems.

CHAPTER V

CHARACTERIZING AND NAVIGATING THE COMPUTE ECOSYSTEM

5.1 Introduction

In the previous chapters, we demonstrated the possibility of clustering mobile devices in order to provide a meaningful and efficient computing service. In addition, we have also showed that, with proper incentives, people are willing to share a portion of the compute resources available in their mobile devices. These findings open the door towards operating a compute service provider that relies mainly on clusters of voluntary mobile devices to share their resources and perform the needed computations. However, identifying the role played by such a compute service provider requires deeper understanding of the full compute ecosystem.

The last few years have seen a tremendous evolution of this compute ecosystem. This evolution has been led by the availability of a variety of cloud-based compute service providers and the deployment of cloud data centers in multiple locations all around the globe. Furthermore, the anticipated deployment of fog and edge computing systems coupled with the potential of having mobile device-based compute service providers add new levels of complexity.

This increased complexity of the compute ecosystem allows mobile device users to have access to a variety of compute options that they can utilize to fulfill their application requirements and meet their target levels of quality of experience. To simplify the mobile device selection decisions, traditional approaches categorized different compute options in two classes (1) low latency edge-compute options, and (2) high latency cloud-compute options. In this chapter, we argue that this is a location dependent view as different compute options can be classified as edge-compute services from certain locations and cloud compute services from other locations. As a result, we argue for a classification-free view of the current compute ecosystem where mobile devices have to be able to learn about the all the the existing compute options and their different network and compute parameters, and and select the ones that maximize their performance.

In this chapter, we use measurement to confirm our insight and shed the light on the current state of the compute ecosystem. In addition, we propose a system that allows mobile devices to better navigate the ever complex compute ecosystem and be able to quickly select which providers are best to execute their jobs. Our system architecture consists of (1) a set of system orchestrators, each of which is responsible for a given geographical area serving its users, (2) a group of mobile agents who either have jobs to execute or volunteer to help the orchestrator, and (3) a variety of compute service providers ready to accept and execute the jobs assigned to them by mobile devices. In our system, each orchestrator relies on mobile agents to acquire data about the existing compute service providers and their connectivity to the different edge networks in the area of interest. Once acquired, this data is analyzed to (1) determine when each of the available providers should be used, and (2) assist mobile agents to make their provider selection decisions efficiently and accurately.

Within our system, we develop a provider selection mechanism designed to minimize the amount of time and state information needed for a mobile agent to make its selection decision. We use a system prototype to validate the accuracy of our selection decision and study the performance of the system. Our results demonstrate the high success rate enjoyed by our provider selection mechanism. Our results also show that the occasional wrong decisions made by our provider selection mechanism has a relatively small impact on the overall system performance introducing no more than 20% computational slowdown.

The rest of this chapter is organized as follows. Section 5.2 provides a measurement-based view of the current compute ecosystem. Section 5.3 discusses the details of our system architecture and presents the main functionalities performed by each of its components. In Section 5.4 we develop our efficient compute-service provider selection mechanism. Finally, we evaluate the performance of our provider selection mechanism in Section 5.5 followed by summarizing the chapter in Section 5.6.

5.2 The Measurement Study

To validate our insights and have better understanding of how the current compute ecosystem look like, we set up a network measurement study in with we use a set of anchor point that exist in various Internet locations to measure the round-trip time between their location and a set of data centers that provide compute services. In our measurement campaign, we have only focused on Amazon EC2 and its various data center locations that are distributed all around the world.

Measurement anchor points: We rely on both GENI [3] and SEATTLE [4] to provide us access to a geographically distributed set of machines in which we install our measurement tools and start our measurement campaign.

Inside Amazon EC2 data centers: To be able to use network measurement tools (like ping) to measure the round-trip time between a given anchor point and a given Amazon EC2 data center, we rented one server instance per data center location and configure it such that it can participate in our measurement campaign.

Acquired data: In our campaign, we have recorded the following data: (1) the location of the data center, (2) the location of the anchor point, (3) the geo-distance for every pair of a data center and an anchor point, (4) the ping round-trip time between every anchor point and every data center instance (mean and standard deviation), (5) the network distance (number of hops) between every data center instance and every anchor point, and (6) the round-trip time between the anchor point and the first hop that can be pinged on the route between it and the data center (mean and standard deviation).

Data Acquisition Mechanism:

- To get the address of the first hop that can be pinged on on the route between the anchor point and the data center instance coupled with the number of hops in the route, we run traceroute once every 30 min for 12 hours to be able to capture any changes to the route due to multi-path load balancing.

- To get the mean and the variance of the round trip time between an anchor point and (1) a data center instance or (2) the first hop in the route, for 12 hours we issue ping requests with frequency 1 ping per second and record the measured round-trip times. Afters the 12 hour measurement period, we analyze the stored data calculating both the mean and the variance of the round trip times.

Gained Insights: Table 9 shows an example of the edge device connections to one of Amazon EC2's data centers. According to the observed latency, some of the devices can classify this data center as an edge compute service provider while others classify it as being a far cloud-based compute service provider. In addition, the table shows that the increase of the physical distance between the edge device and the data center does not imply an increase of the network distance and round-trip time.

5.3 System Architecture

In this section, we provide an overview of our system and its main architectural components depicted in Figure 15. It consists of three main components: (1) *System Orchestrators*, (2) *Mobile Agents*, and (3) *Compute Service Providers*. First, the system orchestrators are running on the cloud and can be deployed as a location based service to enhance the scalability of the system. Each Orchestrator is responsible for

Table 9: Edge device connections to Amazon EC2 data center in Portland, Oregon

Anchor Point	Eugene, Oregon	Vancouver, Canada	Reno, Nevada	Palo Alto California
Physical Dist. (km)	167	420	707	901
Network Dist. (hops)	21	19	20	19
RTT Mean (ms)	11.388	14.107	30.201	25.066
RTT Standard Dev.	3.289	5.155	0.782	0.401
Used Testbed	SEATTLE	SEATTLE	SEATTLE	GENI
Anchor Point	Los Angeles California	Columbia, Missouri	Richardson, Texas	Champaign, Illinois
Physical Dist. (km)	1331	2588	2621	2852
Network Dist. (hops)	16	27	21	24
RTT Mean (ms)	33.617	56.765	58.484	52.032
RTT Standard Dev.	9.195	0.731	0.564	3.614
Used Testbed	SEATTLE	GENI	SEATTLE	GENI
Anchor Point	West Lafayette, Indiana	New Brunswick, New Jersey	Tokyo, Japan	Gudang, China
Physical Dist. (km)	2943	3905	7801	9487
Network Dist. (hops)	24	23	22	37
RTT Mean (ms)	104.010	78.100	144.659	281.822
RTT Standard Dev.	68.282	2.329	70.423	16.404
Used Testbed	SEATTLE	GENI	SEATTLE	SEATTLE

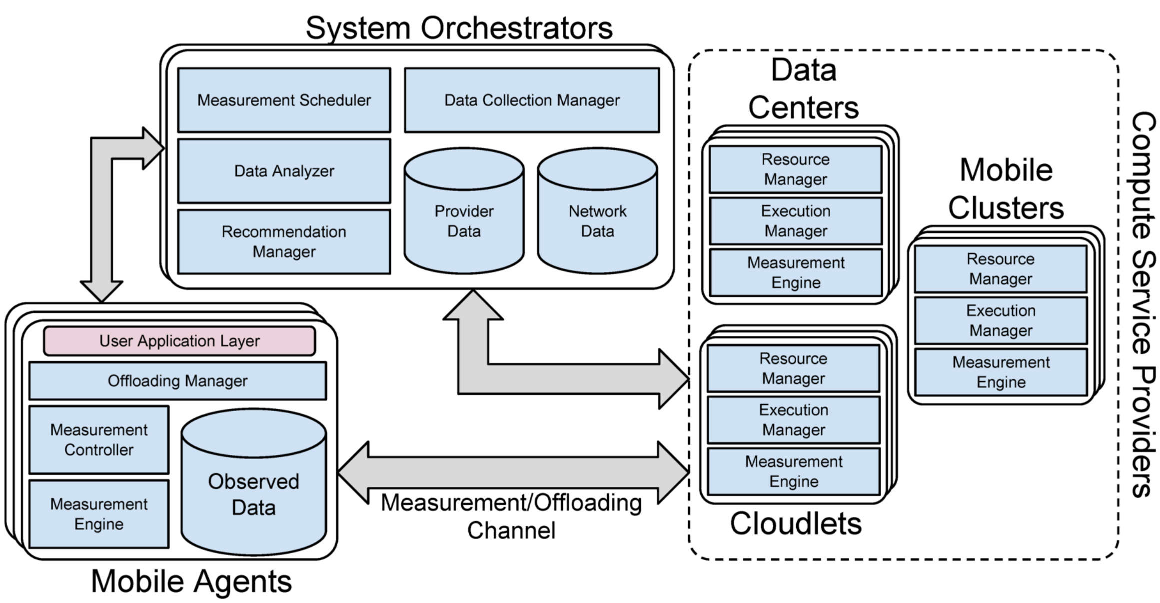


Figure 15: The System Architecture of the Compute Ecosystem Navigator

acquiring the knowledge about a set of compute service providers and how to reach them from different edge networks. It uses the acquired information to help mobile agents better select which compute service provider will meet the requirements of their jobs. The mobile agents, on the other hand, are mobile devices who either have jobs to offload to compute service providers and need the orchestrator's assistance to decide which ones to use or volunteer to help the orchestrator acquire information about their edge network. Finally, compute service providers are responsible for executing the jobs assigned to them by mobile agents. Some of these providers share their runtime information (e.g., job queuing delays) with the orchestrator allowing it to better assist mobile agents in making their assignment decisions.

Similar to Chapter 4, we assume that each job is represented by a directed acyclic graph (DAG) where each node in the graph is a *task* and a directed edge indicates a completion dependency. Each task node is labeled with an estimate of the computational requirement of the task; each edge is labeled with an estimate of the communication requirement to send the task results to downstream tasks that depend on them. This job model covers a wide range of applications, though clearly not all possible jobs. In particular, jobs whose task structure and requirements change at run-time do not fit this model.

We now provide the details of the three main architectural components (depicted in Figure 15).

5.3.1 System Orchestrator

The System Orchestrator is at the heart of our system providing mobile devices with the knowledge of the available compute service providers and the means for making job assignment decision. It maintains data about the different compute service providers as well as their connectivity to edge networks. To be scalable, the System Orchestrator can be deployed as a location based service where different System Orchestrators are deployed each of which is assigned a geographical area that it will keeps its data and services its users.

To acquire the needed knowledge about the existing edge networks and their connections to various compute service providers, the **Measurement Scheduler** determines if it needs to launch an active measurement campaign between certain edge network and a set of compute service providers, selects a set of mobile agents who are willing to perform these measurements, and issues a measurement start request to the selected agents. The **Data Collection Manager** collects the measurement data from mobile agents coupled with runtime data, if shared by compute service providers. This data is stored and then analyzed by the **Data Analyzer** in order to provide guidance to mobile agents with jobs that need to be assigned to one of the available compute service providers. Finally, the **Recommendation Manager** responds to mobile agents queries and provides a recommended list of compute service providers coupled with guidance on how to select which provider to submit a job to.

5.3.2 Mobile Agents

A mobile agent is a client application that operates in two different modes of operations: (1) Job originating mode; and (2) Measurement Volunteering Mode.

In the job originating mode, the **user application layer** issues one or more jobs that has to be performed/executed by one of the compute service providers. For a given job, the **offloading manager** uses job requirements coupled with the knowledge it acquired about various compute service providers and decides which provider that job will be assigned to. Once an assignment decision has been taken, the **Measurement Engine** utilizes the data exchanged (code, inputs, and results) between the mobile agent and the compute service provider to estimate the network parameters (RTT and bandwidth). Furthermore, it also monitors the response time of the jobs and estimates its job queuing delays at the given compute service provider. This information are later shared with the System Orchestrator.

In the measurement volunteering mode, the mobile agent, with the approval of the mobile device owner, is volunteering to help the system orchestrator to acquire information about the the network to which the mobile device is connected to. In this mode, the **Measurement Controller** communicates with the orchestrator to determine the target set of compute network providers that it will actively measure the network parameters while communicating with them. For each target compute service provider, the **Measurement Engine** will measure both the round trip time (RTT) and the achievable throughput, store these measurement in the local measurement database, and share the acquired data with the System Orchestrator.

5.3.3 Compute Service Providers

A compute service provider is responsible for executing jobs assigned to it by mobile devices. The needed compute resources for these providers can be provided by major data centers, voluntary devices at the edge, or any deployed compute infrastructure. For a job that the provider receives, the **Execution Manager** carries the responsibility of fully executing the job and returning the results to the job-originating mobile agent. The **Resource Manager** monitors the resource usage of jobs and gather data about its execution schedule. The data gathered by the resource manager can be voluntarily shared with the system orchestrator in order to enhance the accuracy. Finally, the **Measurement Engine** work with mobile agent to estimate the communication parameters (bandwidth and RTT) between the mobile agent's edge network and the compute service provider.

5.4 Compute Service Provider Selection

The compute service provider selection problem that runs at a mobile agent is very critical to its performance. Typically, a mobile agent must be able to select the compute service provider that suits its demands and will completely execute the the job in hand and return its results as early as possible. This decision, however, has to be taken in a timely manner to avoid introducing additional delays to the job.

Table 10: List of symbols used

Symbol	Description
$\mathcal{B}_k$	The available bandwidth between the mobile agent and the k^{th} compute service provider
$\mathcal{C}_k$	The single thread processing capacity of the k^{th} compute service provider
$\mathcal{P}_k$	The compute level parallelism supported by the k^{th} compute service provider
$\mathcal{T}_{\text{q},k}$	The average queuing delay at the k^{th} compute service provider
c_T	Compute (processing) requirement of task T
o_T	The size of the output of task T
e_T	The size of the executable code coupled with the external data needed by task T
d_{Tk}	Determines whether task T requires the output of task k to start executing (1) or not (0)

5.4.1 Single Task Provider Selection

In this section, we develop a provider selection algorithm that takes a simple job that consists of only one task and selects the fastest compute service provider to fully execute the job and return its results to the mobile agent.

Completion time Model: We first model the job completion time while using certain compute service provider (with index k) as follows:

$$\mathcal{T}_{\text{comp},k} = \mathcal{T}_{\text{in},k} + \mathcal{T}_{\text{q},k} + \mathcal{T}_{\text{ex},k} + \mathcal{T}_{\text{out},k}$$

where $\mathcal{T}_{\text{comp},k}$ is the job completion time using the k^{th} compute service provider, $\mathcal{T}_{\text{in},k}$ is the input transmission time from the mobile agent to the compute service provider, $\mathcal{T}_{\text{ou},k}$ is the output transmission time from the provider to the mobile agent, $\mathcal{T}_{\text{q},k}$ is the job queuing time at the k^{th} compute service provider, and $\mathcal{T}_{\text{ex},k}$ is the job execution time at that provider.

The network delays can be modeled as follows:

$$\mathcal{T}_{\text{in},k} = \frac{e}{\mathcal{B}_k} + \delta_k(e)$$

$$\mathcal{T}_{\text{out},k} = \frac{o}{\mathcal{B}_k} + \delta_k(o)$$

where e is the size of the executable code coupled with the input data of the job, o is the size of the output data, and $\delta(x)$ is a function that captures the additional delays introduced by the data-exchange protocol to send an x amount of data. As a result, the job completion time can be formulated as:

$$\mathcal{T}_{\text{comp},k} = \frac{e+o}{\mathcal{B}_k} + \delta_k(e) + \delta_k(o) + \mathcal{T}_{\text{q},k} + \frac{c}{\mathcal{C}_k} = \frac{e+o}{\mathcal{B}_k} + \frac{c}{\mathcal{C}_k} + \Delta(e,o,k)$$

where c is the compute (processing) requirements of the job, $\mathcal{C}_k$ the single thread processing capacity of the k^{th} compute service provider, and $\Delta(e,o,k) = \delta(e)+\delta(o)+\mathcal{T}_{\text{q},k}$. So the overhead function ($\Delta(e,o,k)$) captures the transmission protocol-specific delays coupled with the provider queuing delay.

Comparing Two Providers: Given two different compute service providers (k_1 and k_2), our goal is determining which of these providers should be used. For the $k_1{}^(\text{th})$ to be selected the following condition must apply:

$$\frac{e+o}{\mathcal{B}_{k_1}} + \frac{c}{\mathcal{C}_{k_1}} + \Delta(e,o,k_1) \leq \frac{e+o}{\mathcal{B}_{k_2}} + \frac{c}{\mathcal{C}_{k_2}} + \Delta(e,o,k_2)$$

This condition can be also formulated as:

$$\frac{c}{e+o} \leq \left[\frac{\mathcal{C}_{k_1}\mathcal{C}_{k_2}\left(\mathcal{B}_{k_1} - \mathcal{B}_{k_2}\right)}{\mathcal{B}_{k_1}\mathcal{B}_{k_2}\left(\mathcal{C}_{k_2} - \mathcal{C}_{k_1}\right)}\right] + \frac{\mathcal{C}_{k_1}\mathcal{C}_{k_2}\left[\Delta(e,o,k_2) - \Delta(e,o,k_1)\right]}{\left(\mathcal{C}_{k_2} - \mathcal{C}_{k_1}\right)(e+o)}$$

Which is equivalent to:

$$\begin{aligned}\frac{c}{e+o} \leq &\left[\frac{\mathcal{C}_{k_1}\mathcal{C}_{k_2}\left(\mathcal{B}_{k_1} - \mathcal{B}_{k_2}\right)}{\mathcal{B}_{k_1}\mathcal{B}_{k_2}\left(\mathcal{C}_{k_2} - \mathcal{C}_{k_1}\right)}\right] \\ &+ \frac{\left(\mathcal{C}_{k_1}\mathcal{C}_{k_2}\left[\mathcal{T}_{qk_2} - \mathcal{T}_{qk_1}\right]\right) + \left(\mathcal{C}_{k_1}\mathcal{C}_{k_2}\left[\delta_{k_2}(e) - \delta_{k_1}(e) + \delta_{k_2}(o) - \delta_{k_1}(o)\right]\right)}{\left(\mathcal{C}_{k_2} - \mathcal{C}_{k_1}\right)(e+o)}\end{aligned} \tag{7}$$

Equation 7 shows that: (1) The provider selection problem can be viewed as a job classification problem where each class of jobs (jobs that have similar compute requirements per unit data) can be assigned to certain compute service providers; (2) This relation is inherently transitive and can be easily generalized to compare more than two compute service providers efficiently; (3) The transitivity of the relation allows for calculating ranges of job's compute per unit data ($\frac{c}{e+o}$) value where a certain compute service provider will outperform the rest; (4) It allows for purging the list of compute service providers and eliminating the ones that will never be selected for jobs coming from certain location/network; (5) Approximate decision boundaries can be pre-calculated and cached by the system orchestrator to help simplifying the selection decisions taken by the mobile agents; (6) Calculating these approximate decision boundaries further helps the system orchestrator to minimize the state information that has to be transmitted to a mobile agent to take its selection decisions; and (7) The accuracy of the approximation increases with the increase of the amount of data transmitted $(e + o)$ allowing the mobile agent to make its selection decision through a simple lookup operation. Finally, note that the majority of the terms in this equation are constants that are known by the system orchestrator which enables the orchestrator to further simplify the adjustment calculations that may have to be performed by the mobile agent.

We highlight that this equation can also be written as:

$$\frac{c}{e+o} - \frac{(\mathcal{C}_{k_1}\mathcal{C}_{k_2}\left[\mathcal{T}_{qk_2} - \mathcal{T}_{qk_1}\right]) + (\mathcal{C}_{k_1}\mathcal{C}_{k_2}\left[\delta_{k_2}(e) - \delta_{k_1}(e) + \delta_{k_2}(o) - \delta_{k_1}(o)\right])}{(\mathcal{C}_{k_2} - \mathcal{C}_{k_1})\,(e+o)} \leq \frac{\mathcal{C}_{k_1}\mathcal{C}_{k_2}\,(\mathcal{B}_{k_1} - \mathcal{B}_{k_2})}{\mathcal{B}_{k_1}\mathcal{B}_{k_2}\,(\mathcal{C}_{k_2} - \mathcal{C}_{k_1})} \quad (8)$$

where the right hand side term represents the boundaries calculated by the system orchestrator and the left hand side terms represent the job-specific calculations that have to be performed by the mobile agent upon having a new job that is ready to be assigned to a compute service provider.

5.4.2 Complex Job Approximation

Despite the clear advantages of our provider selection technique outlined in Section 5.4.1, this technique has been designed to only deal with the simplest form of jobs (a job that consists of a single task).

To be able to use the outlined provider selection process with its efficiency and simplicity while handling jobs with complex task graphs and dependencies, we opt for approximating a given complex job by a single-tasked one. While the job input and output sizes can be directly mapped from the original job model to the approximate one, we use the following equation to calculate the job compute (processing) requirements in the approximate model:

$$c = \left[\sum_{T \in CP} c_T\right] \max\left(1, \frac{\sum_T c_T}{\mathcal{P}_k \sum_{T \in CP} c_T}\right) = \frac{\sum c_T}{\min(\mathcal{P}_k, \frac{\sum c_T}{\sum_{T \in CP} c_T})}$$

where $\sum_{T \in CP} c_T$ is the total processing requirement of the job's critical path, $\sum_T c_T$ is the overall processing requirements of the job, and $\mathcal{P}_k$ is the compute level parallelism supported by the k^{th} compute service provider.

We highlight that this approximation is only suitable for situations were (1) all the compute service providers have the same compute level parallelism, or (2) the compute level parallelism that can be exploited by the job is less than the minimum compute level parallelism provided by the available providers:

$$\forall k; \; \frac{\sum c_T}{\sum_{T \in CP} c_T} \leq \mathcal{P}_k$$

5.5 Performance Evaluation

In this section, we evaluate the performance of our provider selection mechanism and validate the correctness of the decisions it makes. We use a prototype that we implemented on Android and evaluate its performance under two different conditions: (1) Controlled, and (2) In the wild. We start by describing our experimental setup (Section 5.5.1) followed by representative the results from our controlled experiments (Section 5.5.2) and the ones in the wild experiments (Section 5.5.3).

5.5.1 Experimental Setup

We start by describing the experimental job model followed by the summary of the characteristics of the compute service providers we emulate coupled with their network connectivity to the mobile agents. We then summarize our metrics and parameters followed by presenting our baselines.

Job Models: In our experiments, we assume single task jobs. The task's input and output sizes come from a normal distribution with the preset means and standard deviation of 0.25 of the mean. In our experiments, we set the compute requirements of the tasks to match a target value of the compute per unit data.

Provider Model: In our experiments, we assume two compute service providers. To capture the network time versus compute time trade-off, we assume that the first compute service provider has higher compute capacity but connected to the mobile device with a high-latency, low-bandwidth link (typical connectivity with far data centers). The second compute service provider has lower compute capacity but connected to the mobile agent with a high-bandwidth, low-latency link. We will discuss the exact configuration in Sections 5.5.2 and 5.5.3.

We assume that the job queues in both providers are empty ($\mathcal{T}_{q1} = \mathcal{T}_{q2} = 0$). We also assume that all the compute service providers use TCP Reno as their transport layer protocol. Therefore, we rely on Cardwell's model [13] while calculating our additional protocol specific delays ($\delta_k(e)$, and $\delta_k(o)$).

Metrics and Parameters: We are interested in the following performance metrics:

- **Success Rate:** This is the ratio between the number of correct decisions taken by our provider selection mechanism and the total number of decisions. We identify a correct decision as selecting the provider that indeed had the minimum job completion time.

- **Computational Slowdown:** This is the ratio between the job completion time on the selected provider and the minimum job completion times achieved by any compute service provider. This metric is designed to understand the true impact of the incorrect decisions made by our provider selection mechanism.

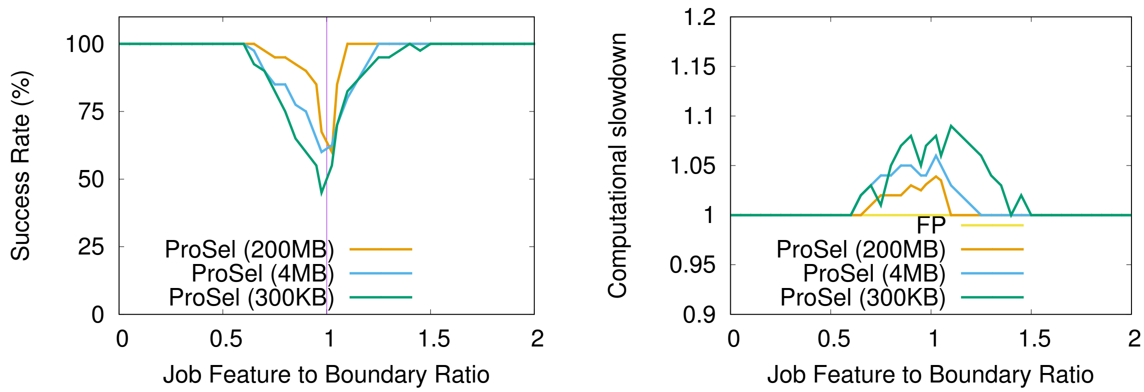
To validate the correctness of the decisions made by our provider selection mechanism, we primarily test against the job feature-to-boundary ratio which is the ratio between the compute per unit data of the job and the boundary at which its decision will be switched. We control this ratio by reverse calculating the corresponding job compute requirements needed to achieve this ratio and setting it to the incoming new job.

Baselines: We compare our provider selection mechanism with the **Fastest Possible (FP)** baseline technique. This approach assumes the ability of assigning the job to all the existing providers and considers a job completed once the fastest of them returns the results. In reality such algorithm might not be possible as the mobile device may not be able to send the job to multiple provider due to cost and energy constraints. Furthermore, the performance achieved by such technique is equivalent to having an oracle that knows, prior to submitting the job, with certainty which provider will finish it earliest.

5.5.2 Controlled Experiment

In this controlled experiment, we run our prototype on three virtual machines one of them represents a mobile agent and the other two represent our compute service providers. We set the compute capacity of the first compute service provider to 100MFLOPS where the compute capacity of the second provider is set to be equal to 50MFLOPS. We use dummynet [2] to emulate the network that connects the mobile agent to each of these compute service providers. We emulate a link with a round trip time of 100ms and bandwidth equal to 512Kbps that connects the mobile agent to the high capacity provider. On the other hand, we connect the mobile agent to the low capacity provider with a link that has 1Mbps bandwidth and 10ms round-trip time.

Figure 16 present the results we acquired while using our controlled setting. The



(a) Success Rate

(b) Computational Slowdown

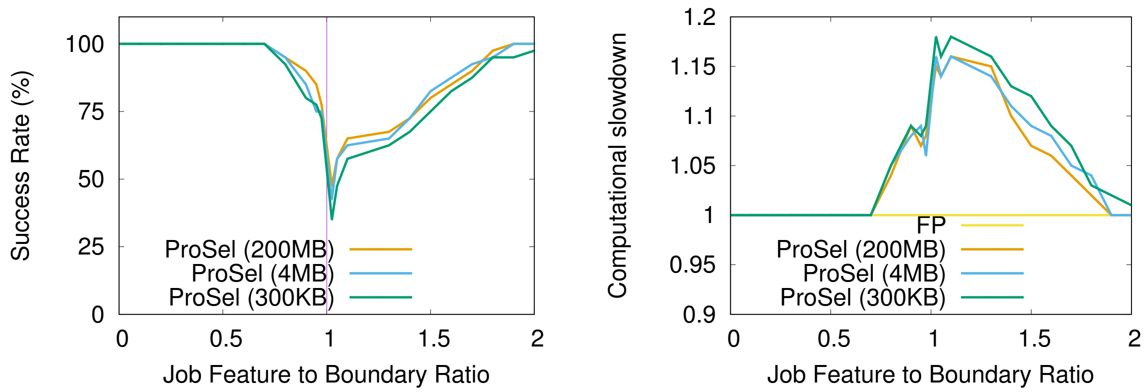
Figure 16: Impact of changing the job feature to boundary ratio in a controlled setting.

figure shows that the farther the task feature is from the decision bounty (>> 1 or << 1), the higher the success rate demonstrating the ability of our provider selection mechanism to make the right selection decision for a very large group of jobs. Although being close to the decision boundary introduced significant amount of wrong decisions, Figure 16(b) shows that the slowdown caused by these decision is less than 10% highlighting that both providers are practically providing the same level of quality of service (completion time) to the job.

Figure 16 highlights the fact that the higher the amount of data associated with the job (input and output), the better the performance of the system (in terms of both success rate and computational slowdown). The reason behind this behavior is that with more data to send and/or receive, the accuracy of our network time estimation increases which leads to more accurate selection decisions.

5.5.3 In the Wild Experiment

In this experiment, we run our prototype on three machines. The mobile agent and the low capacity (50MFLOPS) provider exist in the same WLAN inside Georgia Tech. The high capacity provider (100MFLOPS) however is deployed on Amazon EC2 in



(a) Success Rate

(b) Computational Slowdown

Figure 17: Impact of changing the job feature to boundary ratio in the wild.

Northern California.

Figure 17 presents the results we acquired while using our in the wild setting. The figure confirms the observations we had in our controlled experiments. The reason behind the decreased success rate while going to the Amazon EC2 cloud is due to the fact that the transmission time estimates of sending the input to the cloud and receiving the output are less accurate as they are affected by background traffic. The fact that the slow down is bounded by 20% shows the level of effectiveness achieved by our provider selection mechanism.

5.6 Summary

In this chapter, we presented a system that allows mobile devices to better navigate the ever complex compute ecosystem and quickly select which provider can execute their jobs. We designed the architecture of our system that consists of (1) a set of system orchestrators each of which is responsible for a given geographical area serving its users, (2) a group of mobile agents who either have jobs to execute or volunteer to help the orchestrator, and (3) a variety of compute service providers ready to accept and execute the jobs assigned to them by mobile devices. In our system, each orchestrator relies on mobile agents to acquire data about the existing compute service providers and their connectivity to the different edge networks in the area of interest. Once acquired, this data is analyzed to (1) determine when should each of the available providers be used and (2) assist mobile agents to make their provider selection decisions efficiently and accurately. We used an Andriod prototype that we implemented to evaluate the performance of the system in controlled settings as well as in the wild.

CHAPTER VI

SUMMARY OF CONTRIBUTIONS AND FUTURE WORK

This thesis studies using mobile device clusters as compute platform and takes a set of steps towards realizing a mobile-cluster based compute service provider. The summary of this thesis contributions is:

The FemtoCloud System: This work presents the design, implementation and evaluation of the FemtoCloud system that leverages the available compute resources on a cluster mobile devices to offer compute resource at the edge. It also identifies the importance of the task scheduling problem in this context, formulates the problem within the context of the FemtoCloud system and develops efficient heuristics to solve it.

Workload Management in Edge Femtoclouds: This work presents significant enhancements on the design of the FemtoCloud system allowing it to go beyond supporting a single cluster of mobile devices. The newly proposed architecture is a hybrid edge-cloud architecture that utilizes the cloud for management and to provide a stable service interface while using the edge for low latency computation. In this work we also propose a set of workload management mechanisms that enable an edge computing service comprised of mobile devices with churn to serve DAG structured jobs. We implement a prototype of our system that we use to evaluate the performance of the Femtocloud system. We also use simulations to assess the efficiency of each of our workload management mechanisms, independently. We perform a pilot study to identify suitable incentive mechanisms to encourage users to opt in a Femtocloud system and share their mobile compute resources.

Characterizing and Navigating the Compute Ecosystem: This work provides better understanding of the current compute ecosystem. It presents a a measurement study that characterizes the current state of the compute ecosystem. It proposes the design of a system that allows mobile devices to efficiently navigate the increasingly complex compute ecosystem and efficiently become aware of the existing compute service providers. Within this system, we develop an efficient provider selection mechanism and assess its ability make the right decisions through a set of experiments in a controlled setting and in the wild.

6.1 Future work

There are many directions to extend the work we did in this thesis. We present some of these potential directions below:

- **Running a mobile cluster based compute service provider at scale:** A successful deployment of a mobile cluster based provider with the right set of incentives will eventually lead to more and more people sharing their resources. At this moment, the scalability of the FemtoCloud control architecture will be considered a significant challenge. A single controller model will not be able to handle all the load while deploying enough controllers that are designed to withstand the maximum number of volunteers will be introduce its inefficiencies specially in terms of maintenance cost as well as performance. A promising direction could be designing an elastic FemtoCloud control architecture which is able to grow with the increased number of volunteers and job demand as well as shrink when either of job demand or the volunteering devices decrease.

- **Understanding the economics of the compute ecosystem:** In this thesis, we looked into the current compute ecosystem from the perspective of which provider will be able to better assist a given customer by finishing its jobs earlier. However, there are a lot of other parameters that has to be considered. For example, the cost of using a provider can play a major role on selecting which ones to use. The availability of certain data to be processed (e.g., previous stored information and user data) and the cost of moving it should also be considered. Therefore we further look into the economics and the data usage model in the current compute ecosystem can lead to significant enhancements of the overall user experience and better capturing of her goals and requirements.

REFERENCES

[1] "Boinc: Open-source software for volunteer computing." `https://boinc.berkeley.edu/`. Online; accessed 23-April-2017.

[2] "The dummynet project." `http://info.iet.unipi.it/~luigi/dummynet/`. Online; accessed 13-May-2018.

[3] "Geni." `http://www.geni.net/`. Online; accessed 13-May-2018.

[4] "Seattle: Open peer-to-peer computing." `https://seattle.poly.edu/html/`. Online; accessed 13-May-2018.

[5] Acharya, A. and Badrinath, B., "Checkpointing distributed applications on mobile computers," in *Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on*, pp. 73–80, IEEE, 1994.

[6] Andersen, D. G., Feamster, N., Bauer, S., and Balakrishnan, H., "Topology inference from bgp routing dynamics," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 243–248, ACM, 2002.

[7] Arslan, M. Y., Singh, I., Singh, S., Madhyastha, H. V., Sundaresan, K., and Krishnamurthy, S. V., "Computing while charging: building a distributed computing infrastructure using smartphones," in *ACM CoNEXT*, 2012.

[8] Ashley, P. A., Butler, A. M., ELKeissi, G. M., and Veliyathuparambil, L., "Dynamic security sandboxing based on intruder intent," 2017. US Patent 9,535,731.

[9] Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., and Yang, H., "The case for cyberforaging," in *ACM EW*, 2002.

[10] Beloglazov, A., Abawajy, J., and Buyya, R., "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[11] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S., "Fog computing and its role in the internet of things," in *SIGCOMM MCC*, 2012.

[12] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S., "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, ACM, 2012.

[13] Cardwell, N., Savage, S., and Anderson, T., "Modeling tcp latency," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1742–1751, IEEE, 2000.

[14] Carlen, D., Heck, J., Szilagyi, M., Guis, M., Caruso, K., and Mankin, Y. B., "Fault tolerance for a distributed computing system," May 9 2017. US Patent 9,645,811.

[15] Chen, M., Hao, Y., Li, Y., Lai, C., and Wu, D., "On the computation offloading at ad hoc cloudlet: architecture and service modes," *IEEE Communications Magazine*, 2015.

[16] Choy, S., Wong, B., Simon, G., and Rosenberg, C., "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," in *Proceedings of the 11th annual workshop on network and systems support for games*, p. 2, IEEE Press, 2012.

[17] Chun, B., Ihm, S., Maniatis, P., Naik, M., and Patti, A., "Clonecloud: elastic execution between mobile device and cloud," in *ACM EuroSys*, 2011.

[18] Clinch, S., Harkes, J., Friday, A., Davies, N., and Satyanarayanan, M., "How close is close enough? understanding the role of cloudlets in supporting display appropriation by mobile users," in *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, pp. 122–127, IEEE, 2012.

[19] Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P., "Maui: making smartphones last longer with code offload," in *ACM MobiSys*, 2010.

[20] Devi, C. and Uthariaraj, R., "Load balancing in cloud computing environment using improved weighted round robin algorithm for nonpreemptive dependent tasks," *The Scientific World Journal*, 2016.

[21] Dinh, H. T., Lee, C., Niyato, D., and Wang, P., "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, 2013.

[22] Drolia, U., Martins, R., Tan, J., Chheda, A., Sanghavi, M., Gandhi, R., and Narasimhan, P., "The case for mobile edge-clouds," in *IEEE UIC/ATC*, 2013.

[23] Dusi, M., Fiori, L., and Gringoli, F., "Method and system for checkpointing a global state of a distributed system," June 23 2016. US Patent App. 14/908,131.

[24] Fu, Z., Ren, K., Shu, J., Sun, X., and Huang, F., "Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE TPDS*, 2016.

[25] Gedawy, H., Tariq, S., Mtibaa, A., and Harras, K., "Cumulus: A distributed and flexible computing testbed for edge cloud computational offloading," in *Cloudification of the Internet of Things (CIoT)*, IEEE, 2016.

[26] Gennaro, R., Gentry, C., and Parno, B., "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Annual Cryptology Conference*, Springer, 2010.

[27] Gentry, C. and others, "Fully homomorphic encryption using ideal lattices.," in *STOC*, no. 2009, 2009.

[28] Georgiev, M., Jana, S., and Shmatikov, V., "Rethinking security of web-based system applications," in *ACM WWW*, 2015.

[29] Gordon, M. S., Jamshidi, D., Mahlke, S. A., Mao, M., and Chen, X., "Comet: Code offload by migrating execution transparently.," in *OSDI*, 2012.

[30] Ha, K., Pillai, P., Richter, W., Abe, Y., and Satyanarayanan, M., "Just-in-time provisioning for cyber foraging," in *ACM MobiSys*, 2013.

[31] Habak, K., Shi, C., Zegura, E. W., Harras, K. A., and Ammar, M., "Elastic mobile device clouds: Leveraging mobile devices to provide cloud computing services at the edge," *Fog for 5G and IoT*, 2017.

[32] Keshav, S., "A control-theoretic approach to flow control," *ACM SIGCOMM Computer Communication Review*, 1995.

[33] Kosba, A. E., Papadopoulos, D., Papamanthou, C., Sayed, M. F., Shi, E., and Triandopoulos, N., "Trueset: Faster verifiable set computations.," in *USENIX Security*, 2014.

[34] Kumar, K. and Lu, Y., "Cloud computing for mobile users: Can offloading computation save energy?," *IEEE Computer*, 2010.

[35] Kwok, Y. and Ahmad, I., "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE TPDS*, 1996.

[36] Kwon, Y., Lee, S., Yi, H., Kwon, D., Yang, S., Chun, B., Huang, L., Maniatis, P., Naik, M., and Paek, Y., "Mantis: automatic performance prediction for smartphone applications," in *USENIX ATC*, 2013.

[37] Lewis, G. A., Echeverría, S., Simanta, S., Bradshaw, B., and Root, J., "Cloudlet-based cyber-foraging for mobile systems in resource-constrained edge environments," in *Companion Proceedings of the 36th International Conference on Software Engineering*, pp. 412–415, ACM, 2014.

[38] Li, A., Yang, X., Kandula, S., and Zhang, M., "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 1–14, ACM, 2010.

[39] Liu, Z., Chen, Y., Bash, C., Wierman, A., Gmach, D., Wang, Z., Marwah, M., and Hyser, C., "Renewable and cooling aware workload management for sustainable data centers," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, pp. 175–186, ACM, 2012.

[40] Mtibaa, A., Harras, K. A., and Fahim, A., "Towards computational offloading in mobile device clouds," in *IEEE CloudCom*, 2013.

[41] Mtibaa, A., Snober, M. A., Carelli, A., Beraldi, R., and Alnuweiri, H., "Collaborative mobile-to-mobile computation offloading," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on*, pp. 460–465, IEEE, 2014.

[42] Naehrig, M., Lauter, K., and Vaikuntanathan, V., "Can homomorphic encryption be practical?," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, ACM, 2011.

[43] Navratil, J. and Les Cottrell, R., "Abwe: A practical approach to available bandwidth estimation," in *in Passive and Active Measurement (PAM) Workshop 2003 Proceedings, La Jolla*, Citeseer, 2003.

[44] Nishio, T., Shinkuma, R., Takahashi, T., and Mandayam, N. B., "Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud," in *MobileCloud*, ACM, 2013.

[45] Prasad, R., Dovrolis, C., Murray, M., and Claffy, K., "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE network*, vol. 17, no. 6, pp. 27–35, 2003.

[46] Rodriguez, M. A. and Buyya, R., "A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments," *Concurrency and Computation: Practice and Experience*, 2016.

[47] Saeed, A., Ammar, M., Harras, K. A., and Zegura, E., "Vision: The case for symbiosis in the internet of things," in *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*, ACM, 2015.

[48] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N., "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, 2009.

[49] Satyanarayanan, M., "A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets," *Mobile Computing and Communications Review*, 2015.

[50] Sharma, P., Xu, Z., Banerjee, S., and Lee, S.-J., "Estimating network proximity and latency," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 3, pp. 39–50, 2006.

[51] Sharp, J. A., *Data flow computing: theory and practice*. Intellect Books, 1992.

[52] Sharpe, W. F., Alexander, G. J., and Bailey, J. V., *Investments.* Prentice-Hall Upper Saddle River, NJ, 1999.

[53] Shi, C., Habak, K., Pandurangan, P., Ammar, M., Naik, M., and Zegura, E., "Cosmos: computation offloading as a service for mobile devices," in *ACM MobiHoc*, 2014.

[54] Shi, C., Lakafosis, V., Ammar, M. H., and Zegura, E. W., "Serendipity: enabling remote computing among intermittently connected mobile devices," in *ACM MobiHoc*, 2012.

[55] Shi, C., Ammar, M. H., Zegura, E. W., and Naik, M., "Computing in cirrus clouds: the challenge of intermittent connectivity," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, MCC '12, (New York, NY, USA), pp. 23–28, ACM, 2012.

[56] Shi, C., Pandurangan, P., Ni, K., Yang, J., Ammar, M., Naik, M., and Zegura, E., "Ic-cloud: Computation offloading to an intermittently-connected cloud," tech. rep., Georgia Institute of Technology, 2013.

[57] Singh, R. M., Paul, S., and Kumar, A., "Task scheduling in cloud computing," *International Journal of Computer Scienceand Information Technologies (IJCSIT)*, vol. 5, no. 6, pp. 7940–7944, 2014.

[58] Stojmenovic, I., "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks," in *IEEE ATNAC*, 2014.

[59] Stojmenovic, I. and Wen, S., "The fog computing paradigm: Scenarios and security issues," in *IEEE FedCSIS*, 2014.

[60] Strauss, J., Katabi, D., and Kaashoek, F., "A measurement study of available bandwidth estimation tools," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pp. 39–44, ACM, 2003.

[61] Topcuoglu, H., Hariri, S., and Wu, M., "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE TPDS*, 2002.

[62] Tu, S., Kaashoek, M. F., Madden, S., and Zeldovich, N., "Processing analytical queries over encrypted data," in *Proceedings of the VLDB Endowment*, 2013.

[63] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., and others, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, ACM, 2013.

[64] Wong, K. F. and Franklin, M., "Checkpointing in distributed computing systems," *Journal of parallel and distributed computing*, vol. 35, no. 1, pp. 67–75, 1996.

[65] Wu, F., Wu, Q., and Tan, Y., "Workflow scheduling in cloud: a survey," *The Journal of Supercomputing*, 2015.

[66] Wu, L., Garg, S. K., and Buyya, R., "Sla-based admission control for a software-as-a-service provider in cloud computing environments," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1280–1299, 2012.

[67] Yu, J. and Buyya, R., "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 171–200, 2005.

[68] Zhang, T., Chowdhery, A., Bahl, P., Jamieson, K., and Banerjee, S., "The design and implementation of a wireless video surveillance system," in *MobiCom*, 2015.