Mobile Crowd Sensing in Edge Computing Environment

by

Madhurima Pore

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved October 2019 by the
Graduate Supervisory Committee:

Sandeep Gupta, Chair
Ayan Banerjee
Martin Reisslein
Christophe Cérin

ARIZONA STATE UNIVERSITY

December 2019

ABSTRACT

The mobile crowdsensing (MCS) applications leverage the user data to derive useful information by data-driven evaluation of innovative user contexts and gathering of information at a high data rate. Such access to context-rich data can potentially enable computationally intensive crowd-sourcing applications such as tracking a missing person or capturing a highlight video of an event. Using snippets and pictures captured from multiple mobile phone cameras with specific contexts can improve the data acquired in such applications. These MCS applications require efficient processing and analysis to generate results in real time. A human user, mobile device and their interactions cause a change in context on the mobile device affecting the quality contextual data that is gathered. Usage of MCS data in real-time mobile applications is challenging due to the complex inter-relationship between: a) availability of context, context is available with the mobile phones and not with the cloud, b) cost of data transfer to remote cloud servers, both in terms of communication time and energy, and c) availability of local computational resources on the mobile phone, computation may lead to rapid battery drain or increased response time. The resource-constrained mobile devices need to offload some of their computation.

This thesis proposes ContextAiDe an end-end architecture for data-driven distributed applications aware of human mobile interactions using Edge computing. Edge processing supports real-time applications by reducing communication costs. The goal is to optimize the quality and the cost of acquiring the data using a) modeling and prediction of mobile user contexts, b) efficient strategies of scheduling application tasks on heterogeneous devices including multi-core devices such as GPU c) power-aware scheduling of virtual machine (VM) applications in cloud infrastructure e.g. elastic VMs. ContextAiDe middleware is integrated into the mobile application via Android API. The evaluation consists of overheads and costs analysis in the scenario of “perpetrator tracking” application on the cloud, fog servers, and mobile devices. LifeMap data sets containing actual sensor data traces from mobile devices are used to simulate the application run for large scale evaluation.

DEDICATION

Dedicated to iMPACT lab team, my family, and friends.

ACKNOWLEDGMENTS

Thank you, Dr. Gupta, Ayan for your support throughout my Ph.D. and direction for my Ph.D. I enjoyed being a part of the iMPACT lab. Initially, I received a lot of support from Zahra Abbasi, Georgios Varsamapoulos, which then continued with Ayan Banerjee. It was great working alongside with Vinaya Chakati!

To my lab mates Vinaya, Priyanka, Prajwal, Koosha, Javad, Imane, Subhashish, Shrija, Junghyo, Anurag, Suneet, Ali, Joshua, Robin, Joseph, Bernard, and Ayan thank you for making my time at ASU so enjoyable.

I am thankful for the support in the ways of funding especially by NSF (CNS grant #1218505) and Intel Corporation. I appreciate the team at Intel, Edwin Verplanke, Rashmin Patel and Hari Tadepalli for actively participating in the research project and collaborating with the iMPACT lab team during this period.

I appreciate all the support I got from family back home, my Arizona friends Milind, Ankita and Samir, Parag, my roommates and a big thank you to Rishabh to manage the Seattle home while I was away.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This thesis was developed based on the projects done in the iMPACT lab, ASU. The initial phase of this work was developing solutions for energy efficiency in the domain of data center computing. I acknowledge the support of the iMPACT lab team, Ayan Banerjee, Zahra Abbasi, Georgios Varsamopoulos, Sandeep Gupta and Sayan Kole, in Collaboration with Intel teams. I appreciate input from Intel Advisors Edwin Verplanke, Rashmin Patel and Hari Tadepalli. These projects focused on VM management algorithms and high throughput applications on Many-core devices. Plenty of collaborative work at this time was done in writing proposals, papers, and interactions during these projects. Recent work is focused on developing the edge architecture to deploy MCS applications. In the recent project related to the execution of Smart Grid applications in Edge, I got an opportunity to work with Dr. Anamitra Pal from the Electrical Engineering Department in ASU. I thank Vinaya, Ayan, Dr. Sandeep Gupta and Dr. Anamitra Pal for collaborating and shaping these projects. I appreciate the time taken by my committee members Sandeep Gupta, Ayan Banerjee, Martin Reisslein, Christophe Cérin, and Joseph Hui to evaluate my thesis and provide valuable inputs.

Chapter 1

INTRODUCTION

Mobile Crowd Sensing (MCS) applications use data from mobile users to realize a common goal. These goals are attained by obtaining data from the user environment and processing it in constraints of resources and real-time requirements.

Cloud offload allows the processing of mobile data for many applications. However, the cloud solution may be unsuitable for offloading the MCS data because the mobility of the devices can cause interruptions. Any change in the device context can cause an interruption in the MCS task or acquire data that is not useful. Processing MCS data in the cloud incurs a huge communication cost. Due to these reasons, applications with real-time processing constraints are challenging.

Edge computing can facilitate real-time mobile applications that obtain data from multiple mobile users. It can support real-time analytics required to achieve the goal of the MCS application.

Collaborative MCS applications such as missing child Satyanarayanan (2010); Khan *et al.* (2016); Shi and Jia (2017) use data from mobile users to create a location trace of a missing child. Navigation apps use data from people who are driving to generate navigation instructions for the user Herrera *et al.* (2010); Hull *et al.* (2006). Some of the characteristics of these MCS applications are its location/context awareness, geo-distributed nature, complex analytical processing, acquisition and processing real-time data, and mobility considerations of edge participating devices.

Edge devices have a set of attributes known as context. The values of these attributes can be associated with resources, sensors, user activity, or any system attribute associated with the device or the user environment. Context value on the device determines its suitability to execute a specific MCS task. Context on the user device may change suddenly due to personal use, mobility, exhaustion of resources, or change in environmental conditions. Mobile devices can now evaluate novel contexts such as mental state levels, blood pressure, heart rate levels, user activity, or locations. Multiple contexts can be combined to design interesting MCS applications. For example, obtain the mental and drowsy state of drivers on a specific route or monitor the heart rate of people who are exercising. Context-aware design of the distributed system can enhance the MCS application for the experience of the users, quality of data acquired, resource consumption on the volunteered edge devices, and efficient execution in limited edge resources.

Challenge in executing MCS on edge devices is the uncertainty of context availability involved in mobile edge devices. Also, many MCS applications have complex analytical processing that needs to complete before data can be useful in the application. Processing this data on the edge device or fog enables to generate the required results in real-time.

1.1 Motivation

The use of mobile devices and sensors has grown in recent years up to 50 million devices as shown in Figure 1.1. The increase in IoT applications in various domains has resulted in an unexpected explosion in data (see Figure 1.2). This data explosion is projected to increase due to the high throughput and low latency of 5G technologies. This includes the data acquired in many real-time applications such as traffic monitoring, search, and rescue, disaster monitoring, video surveillance. To obtain relevant data, the strategy is to recruit a large number of users and then execute filter and data analytics to extract it. Data is continuously acquired through various sensors on users. The sensors collect user data that is needed in the MCS app but they

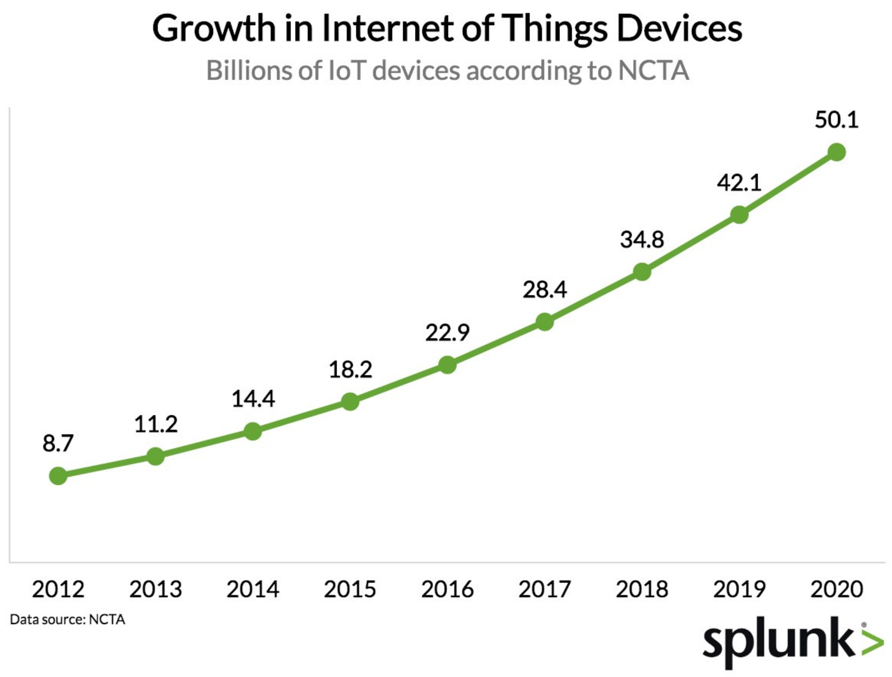


Figure 1.1: Trends in increasing IoT devices.

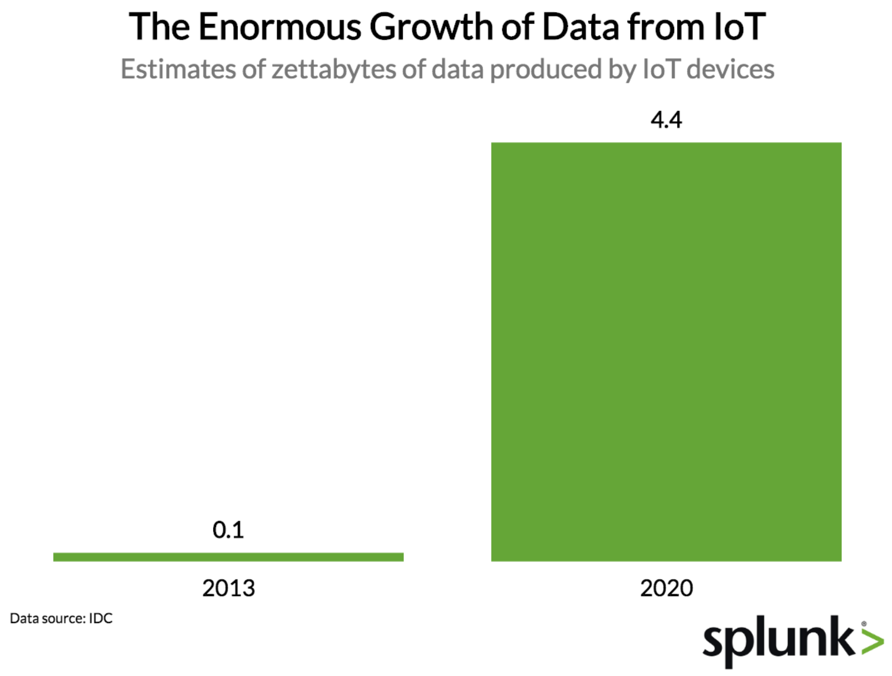


Figure 1.2: IoT data explosion.

also allow evaluation of novel contexts. The advancement in the technology of mobile enables it to support data transfer and analytics locally. Computation advancement supports processing complex analytics by leveraging parallel computational resources on many-core devices. Leading vendors such as Intel and NVidia introduce a multi-core chipset that allows extensively parallel computation that can meet the real-time needs of the application while another option is to utilize the cloud services to accomplish the analytic tasks. However, in case of complex analytics needed for real-time MCS apps, some of the tasks are offloaded on to the nearby edge devices instead of the cloud. Edge computing enables the idea of processing the data near the source of data, thus allows real-time computing. Multi-Access Edge Computing or Mobile Edge Computing (MEC) enables processing at the mobile carrier edge. MEC provides computation resources at the Base station of the carrier to process large mobile data. Edge computing also avoids the unnecessary cost of data transfer, processing the huge data in the cloud. A 2016 Wikibon project about windfarm monitoring over 3 years incurs about 16% cost by reducing by 95% reduction in data using edge processing Mike Wheatley (2016). The results of the Wikibon project show that most of such monitoring projects will be executed using edge devices in the future. This market is currently growing fast at the rate of 32% Kenneth research (2019).

1.2 Overview of Requirements and Challenges

Many MCS applications involve huge processing of data before it can be useful. Several MCS applications that claim real-time operation as seen in Table 1.1. The real-time traffic update application Hu *et al.* (2015) presents one spectrum of real-time operation where the time constraints are relatively tolerant in the order of minutes. On the other end of the spectrum are critical applications such as tracking a person that needs the user data to be processed in order of a few seconds. Many of these

Table 1.1: Characteristics of MCS Applications.

Application Name	Application Features			
	Contextual Requirements	**Sensed Data**	**Computational Requirements**	**Real-time Constraints**
Traffic Monitoring Hu *et al.* (2015)	Location	Accelerometer, GPS (≈30KBps)	Feature classification for stops and driving pattern, traffic forecasting	≈82s (minute) Goh *et al.* (2012)
Event Summarization Chen *et al.* (2016)	Location, Orientation	Images, Videos (≈MB)	Detect event highlights triggered by increased photo capture activity	Post event summary (≈ hours)
Group Event Bao and Roy Choudhury (2010)	Location, Movement, Audio	Images, Videos (≈MB)	Event highlight identification based on audio changes	Post event summary (≈ hours)
Disaster Monitoring Wu *et al.* (2016)	Location, Accelerometer, Gyroscope.	Images (≈MB)	Maximum coverage analysis of destruction	Set by rescue personnel (≈minutes)
Person tracking Shi and Jia (2017), ContextAiDe	Location, Orientation, Accelerometer	Images (≈ few MB)	Estimate next location of person/perpetrator.	(≈few s)

applications involve sending huge data to a central entity like a cloud. A large number of recruited devices create huge data transfer costs. Processing involves a projection of path, statistics on speed and stops and usually can be completed within a few minutes. Some applications like locating a missing child Khan *et al.* (2016) require images and videos from a large number of users. They also involve complex processing to detect and recognize the child to create a timeline for his movement. Analytics are offloaded on the edge devices to locate the missing child. The last known location of the missing child defines the search radius. A critical real-time application such as tracking a perpetrator, on the other hand, involves huge media data to be continuously processed as the perpetrator moves from one location to another. Recently, events of the Boston bombing or 26/11 Mumbai shooting emphasize the active involvement of mobile users that assisted in the capture of the perpetrator/s. The recruited devices need to be selected along the path of the perpetrator and hence have a specific location. The set of user devices can be refined based on the results of the silhouette identification. Thus a new search radius can potentially result in the tracking of the perpetrator. Recently such a tracking application has been introduced by Shi et al. Shi and Jia (2017). This paper discusses the details of computation and algorithm to estimate the location of the user being tracked using MCS. Real-time execution needs the recruitment of context-relevant devices and processing complex analytical tasks on the edge devices during the execution of the MCS app. These sets of edge devices are changing because of changing context due to the mobility of the user and personal usage of devices. Some of the mobile user contexts such as location, connectivity are dependent on the mobility of the user and display patterns Crane (2015). These patterns can be learned and leveraged to select data acquisition devices or execute the processing tasks. Thus an algorithm that uses proactive context information for recruiting a mobile device improves the execution of MCS tasks with a better completion rate and acquiring relevant data.

This thesis proposes an architecture for MCS application design that enables optimized recruitment algorithm considering human interactions on the edge devices.

1.3 Overview of Results and Contributions

The focus of the thesis is the execution of MCS applications on edge devices. Research contributions are summarized towards achieving distributed execution of MCS on the edge devices which address the real-time performance constraints. It also focuses on acquiring context-relevant data from the user devices. Following are the important contributions:

1. *Acquiring Contextual Data:* Propose a model to define contexts and optimize the selection of devices based on the availability of context.

2. *Incorporate Human Mobile Interaction Models:* Human behavior models have to be learned over time to increase their accuracy. But the optimization approaches that incorporate validated models into offload decisions have to be online. ContextAiDe uses stochastic optimization. Other operation parameters considered are response time, mobile device energy consumption, and cost of offload operation with the focus on improving the quality of acquired data.

3. *Processing Efficiently on Edge Devices:* MEC servers and many-core devices execute the real-time analytics tasks. Real-time edge applications are executed using energy-aware mapping schemes that enable energy savings and meet the performance constraints of edge devices.

4. *ContextAiDe Middleware:* MCS middleware architecture is proposed in this thesis. It enables the design of MCS application considering changing context on the edge devices. It consists of APIs to include the common functionality required in MCS and additionally provides customizable modules to monitor and evaluate context, design optimization for user recruitment considering the operational costs, context changes, limited resources, and strict real-time constraints.

1.4 Structure of Thesis

This section elaborates on the detailed structure of the remainder of the thesis and overview of problems and solutions presented in the different chapters. Table 1.2, summarizes the content of the thesis. Chapter 2 discusses important research work in the domains of Edge and Mobile computing. It provides a comparison of ContextAiDe

Table 1.2: Summary of research in design of ContextAiDe architecture.

Context Model	
Problem	The context on the device helps to recruit a device which match the requirement of MCS tasks.
Solution	Chapter 3 provides definitions and metrics which help to meet the requirements of the MCS task Pore *et al.* (2015, 2019).
Context Optimization	
Problem	Recruitment involves optimized selection of devices which match the requirement of MCS tasks.
Solution	Chapter 4 addresses the optimal selection using context optimization. This helps to acquire relevant data and process it efficiently on local or edge devices Pore *et al.* (2019).
Human Interaction Models	
Problem	Leveraging proactive context for scheduling MCS tasks using context prediction
Solution	Chapter 5 provides a novel strategy to predict the mobile user contexts Pore *et al.* (2015). Different time series models and machine learning models are use for prediction of various parameters Abbasi *et al.* (2013b,a).
Many-core Devices	
Problem	Energy aware scheduling on Many-core devices
Solution	Chapter 6 uses an interference model which enables energy aware scheduling of applications on many-core devices Pore *et al.* (2013, 2014).
VM colocation	
Problem	Energy aware scheduling of colocated VMs
Solution	Interference of VMs and contention of resources causes performance degradation. Chapter 6 proposes an interference model for scheduling VMs in a data center Pore *et al.* (2012). Further, the interference model is used to map the applications on servers in a data center environment.
Elastic VMs	
Problem	Scaling VMs based on VM workload in OpenStack
Solution	In mobile edge computing, mobile application run on VMs in Cloud-Radio Access Network (CRAN). These VMs exhibit workload patterns that repeat on daily and weekly basis. VM workloads models are discussed in Chapter 5. The models are used to developed automated VM scaling algorithm which scales up and down described in Chapter 7.
Evaluation of ContextAiDe	
Problem	Evaluation of ContextAiDe in user environment
Solution	MCS application, such as perpetrator tracking are build using ContextAiDe API. Evaluation is performed by executing the application on limited Android devices available in the iMPACT lab. But large scale evaluation is performed using simulation of traces extracted from the LifeMap Data Set Kotz *et al.* (2009). Evaluation set up and results are discussed in Chapter 8 and Pore *et al.* (2019).

architecture with other middleware architectures. Chapter 3 discusses the background of MCS applications, and edge components. It further explains the system model and provides the definitions and metrics, which are used to evaluate context matching of a recruited device. Stochastic optimization is presented in Chapter 4. This chapter presents the details and design of user recruitment and the ContextAiDe architecture components. Standard definitions of through API enable the MCS tasks such as, device discovery, context monitoring and evaluation, optimization, task execution. Prior information about the user context and device context helps to select the user devices in ContextAiDe. Chapter 5 explains the details of human interaction models. These models help to predict the context which is used in context optimized recruitment. It is also necessary to execute the MCS tasks in real-time constraints. Some of the analytics and computational tasks are offloaded to cloud or edge data centers. Chapter 6 describes details of performance and energy-aware scheduling on many-core devices and servers. Many MCS applications are executed in MEC. Chapter 7 discusses the elastic VM management algorithm, which scales the VM according to user workload. Chapter 8 shows the details of evaluation of ContextAiDe architecture. Conclusion and future directions of this thesis are presented in Chapter 9.

Chapter 2

RELATED WORKS

Edge computing enables the execution of MCS applications in a distributed environment in a way that addresses the real-time requirements of the application. In this chapter, presents some of the prominent research works in the areas of MCS, fog and edge computing, MCS middleware, and recent trends in context-aware mobile computing.

2.1 Mobile Crowd Sensing

MCS acquires data from a large number of mobile users spread over a wide sensing area. MCS can potentially enhance services in different application areas such as safety and emergency Sun *et al.* (2011); Mokryn *et al.* (2012); Khan *et al.* (2016); Simoens *et al.* (2013), traffic estimation Panichpapiboon and Leakkaw (2017) as well as entertainment Toledano *et al.* (2013). Participatory sensing and opportunistic sensing are two main types of MCS applications. Participatory sensing requires active user involvement, such as Waze Hardawar (2012). Opportunistic sensing gathers information from sensors and mobile devices without any user involvement Mokryn *et al.* (2012). Recent works Meurisch *et al.* (2013); Xiang *et al.* (2017) discuss large-scale monitoring of geographically distributed mobile devices. Challenge is to improve the energy consumption of user devices by adopting techniques such as piggybacked sensing Bradai *et al.* (2016) or sensing with context preference Capponi *et al.* (2017). ContextAiDe objective is the immediate execution of tasks considering the real-time requirements of MCS.

2.2 Fog and Edge Computing

Fog computing is a solution for providing fast computation using edge devices. Fog computing supports sense-process-actuate sequence or stream processing for enhanced data analytics. Fog devices allow processing near the data source and hence avoid unnecessary communication costs. It enables gathering intelligence in critical situations such as disaster monitoring Higashino *et al.* (2017). Different paradigms of edge computing exist, such as Mobile Edge Computing (MEC) Tran *et al.* (2017); Sapienza *et al.* (2016). In MEC, the operators of mobile companies host applications near the end-users. There is a growth of smaller cloud units such as Micro-Clouds Wang *et al.* (2017b), which are located near the end-user and share the computational tasks with the edge devices. This research focuses on two aspects of fog and edge computing: 1.) Challenges associated with dynamically built peer-to-peer networks on edge; 2.) Devise scheduling schemes for energy saving and fast execution on fog/edge devices while running multiple server applications to support mobile apps.

2.3 MCS Middleware

The middleware for MCS enables easy development of MCS applications. It enables easy integration of the generic functionalities, which are essential for most MCS applications, thus shortening the development time. It also allows extending common API and definitions to include App-specific functionalities. MCS Middleware have been broadly classified based on size of MCS such as group Miluzzo *et al.* (2007), community Jayaraman *et al.* (2013) or urban Sherchan *et al.* (2012). Initial MCS versions include POGO Brouwers and Langendoen (2012) and Medusa Ra *et al.* (2012), which offload sensing tasks onto the user's mobile devices. Medusa also provides an elaborate service-based middleware with a language called Medscript with features such as task

Table 2.1: MCS Middleware Features.

Research work	**User Recruitment**	**Local Analytics**	**Context Availability**	**Uncertainty**	**Anticipatory Context**
McSense Cardone *et al.* (2013)	✓				✓
Medusa Ra *et al.* (2012)		✓			
CAROMM Sherchan *et al.* (2012)	✓				
MOSDEN Jayaraman *et al.* (2013)	✓	✓			
CARDAP Jayaraman *et al.* (2015)	✓	✓			
SPACE-TA Wang *et al.* (2017a)	✓	✓			
DSE Fiandrino *et al.* (2017)	✓	✓	✓		
A3Droid Baresi *et al.* (2016)	✓	✓	✓		
Re-OPSEC Bradai *et al.* (2016)	✓			✓	✓
BLISS Han *et al.* (2016)	✓			✓	✓
ContextAiDe	✓	✓	✓	✓	✓

monitoring and execution management. HoneyBee adopts a load balancing method to reduce the computational burden on a single device Fernando *et al.* (2012). MOSDEN attempts to design middleware for opportunistic MCS, which includes data sensing, collection, and storage as independent modules making it reusable for different sensing applications. Additionally, they use plugin environments for application-specific sensor requirements Jayaraman *et al.* (2013, 2015). CAROMM uses the contextual information to extract relevant data from large-scale data acquired Sherchan *et al.* (2012). A3Droid Baresi *et al.* (2016) proposes user management, data acquisition strategies, failure recovery, and remote code execution platform. ContextAiDe uses similar methods for execution and failure recovery, but context optimized user recruitment is an important consideration in ContextAiDe. Table 2.1 presents a comparison of recent middleware architectures with ContextAiDe.

User Recruitment: The user recruitment strategy addresses the goals of MCS, such as real-time processing, energy-saving, improving the quality of data acquired, or processing geographically distributed data. An important consideration in environmental monitoring MCS is the space-time distributed acquisition strategy. A new

DSE (Distance, Sociability, Energy) aware recruitment policy is proposed for distributed MCS with large time constraints Fiandrino *et al.* (2017). To obtain more relevant data, Reddy et al. suggest a recruitment strategy that employs user activity in addition to the location and time Reddy *et al.* (2010) or evaluate multiple contexts on a device using CATA scheme Hassani *et al.* (2015). ContextAiDe proactively recruits devices based on their expected contexts and iteratively revises selection with changes in context on devices, and updated context requirement as the task execution is in progress.

2.4 Context Awareness and Context Estimation

Sensor data acquired in MCS can be used to obtain valuable information or the user with the help of contexts. Inclusion of innovative contexts, such as SafeDrive, NeuroMovie Pore *et al.* (2015); Oskooyee *et al.* (2016, 2015) widens the scope of context-aware computing . The authors propose methodologies to derive the physiological state of the user from brain sensor data and use it tp develop interactive MCS applications. Another such interesting application is creating video highlights of group meetings, called Movi Bao and Roy Choudhury (2010). This automated application annotated the interesting events using the context information of the people in the group. Factors such as mobility, network connectivity, and personal usage can dynamically change the context of the mobile device. Learning the patterns of mobility, and behavior can enhance the performance of mobile applications Garcia Lopez *et al.* (2015). Some works have suggested the use of mobility models, which can be used to estimate the context changes Banerjee and Gupta (2015); Lee and Shin (2013). Oracon learns the context patterns online using several techniques, such as semi Markov models and alignment of patterns, that are used to estimate the user context da Rosa *et al.* (2016). Prediction of the user's location is also performed using Spatiotemporal HMM models Lv *et al.* (2017). Mobile usage is associated with uncertainty caused due to mobility and device usage. Researchers consider the uncertainty in mobile device contexts in some of the newer user recruitment strategies Han *et al.* (2016). This strategy deals with limited incented budgets in the uncertain mobile user environment. Using compressive sensing combined with intra-data correlations, active and transfer learning methods to reduce the processing needs of the edge devices is proposed in SPACE-TA. Wang *et al.* (2017a).

ContextAiDe opportunistically discovers suitable candidate devices that improve the quality of contextual data acquired using context optimization algorithm. In case of failure of task due to context unavailability, this architecture uses two-stage stochastic optimization that is useful for two reasons: 1. Proactive offload strategy that optimally selects devices based on estimated context; 2. A reactive strategy to handle the unexpected change of device contexts and enables them to continue the execution of tasks.

Chapter 3

SYSTEM MODEL AND ASSUMPTIONS

This chapter presents the components of edge architecture and definitions that enable the design of the MCS application. It presents a formal definition for context concerning the requirements of an MCS application and context available on the devices that can potentially execute the MCS applications.

3.1 System Model of Edge Architecture

Devices in the edge architecture can be broadly classified into five types, as shown in Figure 3.1. Sensors and actuators are the device interfaces to the user environment. Mobile devices are connected to the sensors and have a communication link to the other layers in the architecture. Fog and MEC servers are processing the analytical tasks near the edge, while Cloud provides additional resources.

a) Embedded sensors or actuators collect data from the user environment and perform actuations. These devices are physically installed on users and transfer data to the nearby device using builtin communication capability. They may also have limited computing capability.

b) Personal devices are an intermediate data hub for sensor and actuator data or a computation device. MCS data acquired from the users is either processed locally on personal mobile devices or sent to other devices. The mobile devices have analytical processing capability as well as support the offload of the MCS task. However, inherent mobile nature and personal usage are a challenge in using the edge devices.

c) Fog servers are located near the edge. Fog server enables the computation of MCS tasks near the data source. They exceed in computational capabilities, and storage

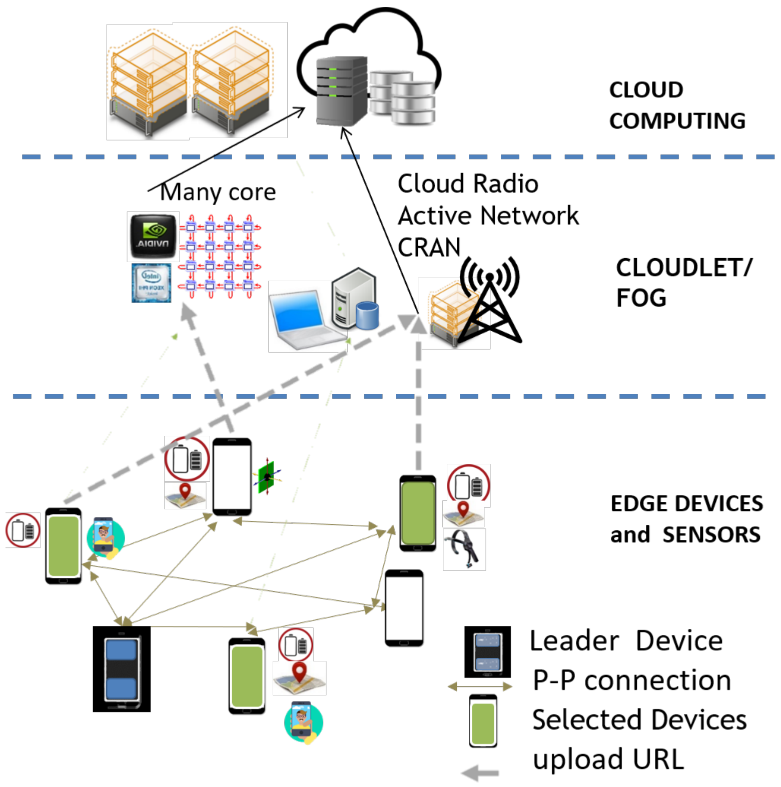


Figure 3.1: System Model Showing Components of Edge Computing Infrastructure.

capabilities, compared to the personal devices. Fog servers reduce communication costs supporting real-time applications.

d) Cloudlets are situated between the edge and cloud and provide quick compute services for MCS tasks. They are a small data center in a box with additional compute and storage resources.

e) Cloud Radio Access Network (CRAN) enables the mobile data to be processed at the edge of the mobile network. MCS application data is processed in CRAN to address the low latency, interactiveness, and real-time automation needs. CRAN processes the IoT data, locally and in a private environment near the edge, instead of sending the data upstream to be processed in the cloud. To support the mobility of edge users, the CRAN involves virtualization of resources and also network components. CRAN management is discussed in Chapter 7 and Appendix C.

f) Cloud has the most computational, communication, and storage capability in the hierarchy. These cloud services are capable of scaling their resources and can support multiple numbers of application VMs. The cloud servers are usually associated with a cost per usage.

Some of these servers may be multicore devices that support the execution of computationally heavy applications.

3.2 Standard and Protocols

Edge architecture uses IoT standards and protocols to establish communication and interoperability between the different components. Broadly these protocols are divided into two main categories: Data Protocols; and Network protocols (see Appendix A for details). In ContextAiDe, mobile devices and sensors are a part of IoT. ContextAiDe API has built-in support for some common sensors e.g. GPS for location data, accelerometer, etc.

ContextAiDe uses the WiFi protocol to communicate between edge devices and any offload operations. ContextAiDe uses PubNub, a publish-subscribe API and TCP sockets, based on HTTP network protocol to establish the network of devices.

3.3 Application Requirements

Edge architecture can support the execution of different types of applications. Geo-distributed sensing involves composing inferences from data acquired in various locations, e.g., oil plants, real-time traffic management, and smart grids. The spatiotemporal requirements of these applications use location and time context of sensed data. Real-time applications such as virtual reality, video security are increasingly complex applications that benefit from the edge architecture. Low latency and efficient processing enable the real-time response of these applications. Attributes of data acquired such as location, time, and size of data are useful to assign computation on the edge devices. Smart and connected applications such as real-time navigation and live news reporting, use data acquired from multiple mobile devices. The data is useful if it can update information along a certain navigation route, or in an area associated with the news event.

3.4 Definitions for Evaluation of Context

The MCS application requirements indicate specific attributes of the device, useful for executing the task. In the runtime, the values of device attributes are evaluated to find a suitable device. The "context" is defined as follows:

Definition 1. ***Context:*** *Context is a key-value pair $\{\psi, C\}$ where ψ is an attribute of the mobile device, and C is a value taken by the attribute ψ on the device.*

Context is defined to be exact or preferred. The idea behind using such a proposed distinction is that some application context requirements may be hard and need to be matched by the device exactly. Some requirements may be soft and matched by the device within an acceptable range. The exact context is denoted by $\{\psi^{\chi}, C^{\chi}\}$ while, the preferred context is denoted by $\{\psi^{p}, C^{p}\}$. Ω is a set of multiple contexts associated with a device. Certain contexts in this set Ω may have a regular usage pattern (Ω'). Others contexts may not exhibit such usage patterns, and be dependent on factors such as environmental conditions or sudden user activity.

The user visits a certain location in a daily or weekly routine. These visits occur at a specific time of the day or repeat on certain days of the week. Similarly, the routine behavior of users can create a pattern in charging and usage of a device. These different patterns can be learned and utilized in deciding MCS user recruitment. Context distance is used to evaluate the suitability of a device to execute the MCS task.

Definition 2. ***Context Distance:*** *For a given attribute ψ context distance is the difference between the context values C_d on the surrogate device and requirement C_r and is represented using a function $\delta(.,.)$ that takes the context values as input and outputs a real number. Context distance for the exact and preferred contexts are given as follows:*
1. $\delta(C_d^X, C_r^X) = 0$ *for exact context and*
2. $0 \leq \delta(C_d^p, C_r^p) \leq \epsilon$ *for preferred context, where ϵ is an application specific parameter decided for the accurate operation of the application.*

The context-matching index normalizes each context and establishes a standard way to evaluate multiple contexts on a device.

Definition 3. ***Context Matching Index:*** *Context Matching Index $\mathcal{I}$ is a ratio of context distance of a given preferred context to the acceptable deviation of ϵ.*

$$\mathcal{I} = \frac{\delta(C_d^p, C_r^p)}{\epsilon}. \tag{3.1}$$

Context Set Ω represents the set of contexts associated with a device. The context on the devices consists of resources, usage dependent, physiology, and activity. Some of these contexts vary according to user behavior patterns, e.g., location and data usage.

An app may have multiple context requirements. Let the importance of each requirement is denoted by weight, w_k. Context Sense Index is used to compare contexts on multiple devices as follows:

Definition 4. ***Context Sense Index:*** *Context Sense Index for a mobile device is defined as the sum of weighted context matching index for each preferred context C_d^p*

in Ω_d.

$$\Upsilon_d = \sum_{k=1}^{K} w_k.\mathcal{I}_k, \tag{3.2}$$

w_k is the weight associated with the context $\{\psi_k^p, C_k^p\}$ and is used to set priorities for the preferred context. Sum of the weights for a given context set is 1. The app developer sets the weight w_k for each C_k^p.

The definitions in this chapter enable the selection of a device based on available context to meet the requirements of the application. However, the mobility and usage of a device can cause dynamic context changes and interrupt the MCS app execution. The challenge is to model the human interaction patterns and use them for context optimized user recruitment. The proactive strategy will ensure the availability of context during execution.

Chapter 4

USER RECRUITMENT ARCHITECTURE

MCS utilizes ad-hoc devices to execute MCS tasks, hence the context availability on the devices is significantly uncertain. Sometimes mobility or personal usage may cause loss of context required for MCS task execution. Moreover, the context requirement of the MCS task may change in real-time. This chapter details the selection of user devices to reliably execute MCS tasks with the additional overhead of context changes. It also provides ContextAiDe architecture, consisting of the components which define contexts, user selection strategies, and middleware components in MCS.

4.1 User Recruitment Algorithm

Mobile devices available to perform the MCS tasks are denoted by N and represented as Eq. 4.1. A device x_i of set N may or may not be selected by the ContextAiDe based on the availability of the required context.

$$N = \{x_1, x_2, x_3, \ldots, x_N\} |\; x_i \in \{0, 1\}. \tag{4.1}$$

An MCS application has K context requirements and the context requirement set Ω_r is given by $\{\{\psi_1, C_1\},\{\psi_2, C_2\}...\{\psi_K, C_K\}\}$. Some of these requirements can be *hard* for exact contexts or *soft* for preferred contexts. Important contributions of ContextAiDe such as user recruitment is designed based on context availability.

4.1.1 Context Optimization Problem

The objective is to select surrogates with minimal context sensing index within real-time constraint as well as resource usage limits of devices. In the first step, given a request for S surrogates that meet the context requirements Ω_R, the surrogate selection approach first selects a set $S_H > S$ surrogates each of which match the exact context, i.e., $\delta(C^{\chi}_{x_i}, C^{\chi}_{r}) = 0, \forall x_i \in S_H$. In the second step, S_H is further refined using two-stage stochastic optimization.

Stage 1: The first stage determines the optimized set of surrogates with minimal context deviation by using stochastic models of the historical trends for a resource and operational overheads. The context set Ω_R comprises of expected values for some of the contexts that are dependent on the usage and mobility of the device. Expected values are computed using several stochastic models discussed in detail in Chapter 5. These models reduce the failure of execution due to lost contexts, such as mobility, network connectivity, and battery level on the device.

Optimization algorithm assumes that device selection is associated with a penalty P_i and is given by Eq. 4.2.

$$\begin{aligned} P_i &= \Upsilon'_i \;|\{\; \delta(E[m_i(\psi^p_i)], C^p_r) \leq \epsilon \,\forall\, C \; in \; \Omega\}, \\ &= P \gg 1 \; Otherwise. \end{aligned} \tag{4.2}$$

Here $m_i(\psi^p_i)$ is the stochastic model of the context $\{\psi^p_i, C^p_i\}$ in terms of the attributes based on historical usage or mobility data. $E[m_i(\psi^p_i)]$ and C^p_r in Eq. 4.2 are the expected and required value of context on device i. Υ'_i is computed using the expected values of the preferred contexts, using Equation 3.2 as,

$$\Upsilon'_i = \sum_{k=1}^{K} w_k \mathcal{I}'_k, \;\; \mathcal{I}'_k = \frac{\delta(E[m_i(\psi^p_i)], C^p_r)}{\epsilon}. \tag{4.3}$$

The formulation for context optimized user selection is designed based on penalty:

$$min \sum_{i=0}^{N} x_i.P_i, \qquad x_i \in \{0,1\} \tag{4.4}$$

Subject to

$$\sum_{i=0}^{N} x_i \geq S$$

$$\tau_i^s(t) + \tau_i^t(t) < \tau^T$$

$$R_i(t) - R_i^{\Theta}(t) > R_i^T.$$

where P_i is the penalty associated with the selection of device as given in Eq. 4.2, S is the number of devices required for assigning the sensing task. $\tau_i^s(t)$ is time taken for data sensing and $\tau_i^t(t)$ is the data transfer time and τ^T is the execution time constraint. Both $\tau_i^s(t)$ and $\tau_i^t(t)$ represent the time varying operational overheads. Such overheads also vary for different surrogates and stochastic models have to be used in order to estimate such details. ContextAiDe uses a data driven approach towards modeling such delays and is application specific. This aspect is discussed in further detail in Section 8.1. R_i is the current resource availability and R_i^{Θ} is the amount of resource usage on the surrogate device. R_i^T is the limit set on the resource usage by the device owner. R_i^{Θ} is obtained from observing the previous usage history of the person. Stochastic models for R_i^{Θ} is discussed in more detail in Chapter 5.

Stage 2: The second stage refines the optimization decision of stage 1 based on the actual context values available on the devices for all the context attributes in Ω. The objective function is the same as Stage 1. However, during this stage, the optimized set of stage 1 is validated with the current context values on the selected set of devices. If all the devices in the selected set are within the acceptable context deviation, then there is no task reassignment to avoid any additional overhead. But if one or more devices of the optimized set are no longer in the acceptable context deviation range, then the architecture tries to find the new set of devices within an objective of minimal context deviation. The optimization process considers set $N'{=}N - x'$ to obtain the new devices. The number of devices required in new optimization is $S - \Sigma x'$. Unlike stage 1 of optimization, which uses expected values for the context variables, stage 2 uses the actual context values. That is, in stage 2, context distance is calculated as $\delta(C_i^p, C_r)$. This stage selects a newly optimized set of surrogates S'_H from the set S_H using the current values of context for assigning the sensing task.

Algorithm 1 User Recruitment Algorithm

```
 1: procedure RecruitDevices($no, C_r$)
 2:     Discover nearby devices
 3:     Read $C_x$                                          ▷ Device $x$ context
 4:     if $\delta(C_x^x, C_r^x) = 0$ then
 5:         Add device to $S_H$
 6:     end if
 7:     while $S^* \leq S$ do                                ▷ Stage 1
 8:         Compute $P_x$.          ▷ Penalty P using estimated values of $C_x^p$
 9:         Minimize $P_x$.                          ▷ S is requested devices.
10:     end while
11:     while $\delta(C_x^p, C_r^p) \geq \epsilon$ do                    ▷ Stage 2
12:         Compute $P_x$.                        ▷ Using actual values $C_x^p$
13:         Minimize $P_x$.                                  ▷ revise set S
14:     end while                              ▷ repeat until task completes
15: end procedure
```

The penalty for a device in stage 2 optimization is given by:

$$P_i = \Upsilon_i \; |\{ \; \delta(C_i^p, C_r) \leq \epsilon \; \forall \; C \; in \; \Omega\},$$
$$= P \gg 1 \; Otherwise. \tag{4.5}$$

and the objective function is given by

$$min \sum_{i=0}^{N'} x_i.P_i, \mid x_i \in \{0, 1\} \tag{4.6}$$

where x_i is a device selected for MCS task and P_i is penalty associated with device i.

The process of recruitment can be explained using the Algorithm 1.

User-defined preferences: The ContextAiDe architecture allows a user to set limits on their device resource that can be used by the MCS task. These preferences can be **bounds**, e.g., A user might only want to volunteer their device if the device state of charge is not reduced by more than 15%, as suggested in recent studies Heng Zhang (2017).

4.1.2 Complexity of Optimization

The MCS app specifies K contexts required to execute the task on a certain device. ContextAiDe monitors these contexts on the participating devices. Devices with the exact context are filtered. For N devices, the complexity of optimization is the sum of computing the CSI index for each device and sorting the N devices according to their CSI indexes. The computation complexity is:

$$O(K) + O(NlogN) \tag{4.7}$$

4.2 ContextAiDe Architecture

ContextAiDe defines a standard way to build MCS apps. This distributed architecture allows us to connect, and execute code on user devices, by standardizing the underlying communication, and execution. ContextAiDe has four components (Refer Figure C.2):

4.2.1 Distributed Platform

ContextAiDe provides a distributed execution platform. Pre-decided parts of the application execute on selected user devices and the edge servers. The MCS application task include the task of data acquisition, preprocessing the data and analytical processing of the data.

4.2.2 Middleware

ContextAiDe *Middleware* provides implementation of common functionalities required by most of the MCS apps. These functions are in the form of API, and their details are described below:

Device discovery: This component connects with all the available devices and communicates with them. The devices communicate on a publish-subscribe channel. This module communicates the App requests and codes to selected devices.

User recruitment: It refines and recruits the devices selected by optimizing the preferred context requirements of the MCS app under the device resource constraints. Change in the availability of contexts on devices executing the task updates the recruitment decision.

Continuous context monitoring: It monitors contexts of volunteer devices at the desired interval and maintains history. During the execution of MCS tasks, it re-evaluates context availability and initiates the recruitment process if required.

Failure handling: It monitors the connection of participating devices. It also monitors the status of offloaded tasks. This module notifies the leader if any of the offload tasks fail or a device disconnects while executing the task.

Communication Management: It is responsible for establishing and maintaining

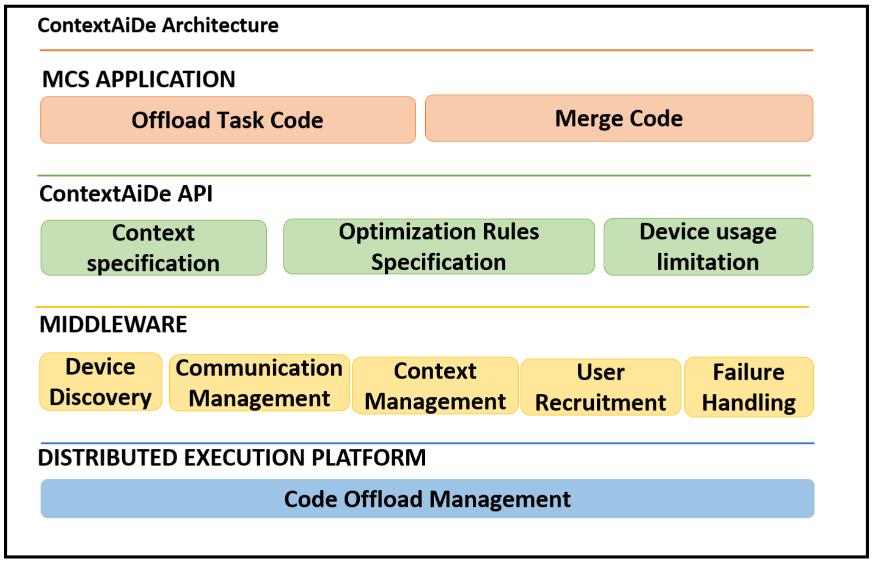


Figure 4.1: ContextAiDe Architecture.

the communication link between different user devices, cloud, or the edge devices through various mechanisms including WiFi, WiFi Direct, Bluetooth, or 4G.

4.2.3 API

ContextAiDe API is a programming interface. A developer can specify MCS application context requirements both hard (*exact*) and soft constraints (*preferred*), device resource limitations, and parameters of the optimization processes (used in user recruitment), context prediction and monitoring, real-time MCS application performance.

4.2.4 Application

This module defines a way for a developer to write the code for each MCS task. This code is executed on the user devices running Android OS.

Chapter 5

HUMAN MOBILE INTERACTION MODELS

Multiple sensors on a mobile device obtain data locally that is used to derive human interaction models. This data is recorded at regular time intervals on the device and used to predict the value in the next time interval. This prediction may require diverse mathematical approaches such as time series prediction, machine learning techniques, etc.

MCS applications generate traces of data that are usually characterized by some usage or behavior patterns. Examples of such data can be electricity prices, generation of solar power w.r.t. smart grid as well as user location or battery usage w.r.t. mobile applications. Different seasonality or correlation patterns may be exhibited in such data.

5.1 Time Series based Predictive Modeling

Predictive modeling uses mathematical models and computational methods to predict the outcome of an event, or a phenomenon. Computational models involve simulation of the black-box model e.g. neural networks or bagging tree-based models to predict the credit rating of a borrower.

The simpler mathematical model involves regression or machine learning approaches to create a prediction model. It involves similar processing steps: cleaning data for outliers and missing values, formatting, a subset of data for training, validation and testing, training model parameters, error measurement of validation and test sets, calibration, fine-tune model parameters for better accuracy and model deployment.

Time Series Model

A stationary time series has no trend and has a constant variance over time. A most common model to analyze such stationary data with seasonality patterns is the Time Series Model. The time series model involves Auto-regressive and Moving Average terms that can describe the seasonality of the data. The AR and MA terms can be determined by looking at the Autocorrelation (correlation of time series by a lag) and Partial Autocorrelation plots (correlation results after removing any correlation due to terms at shorter lag).

Determining AR Terms

ACF plots help to determine the Autoregressive terms. ACF plots are strong up to lag of k and trails off after subsequent lags after the effect of AR terms diminishes. As PACF indicates direct relation between k and its lower terms, the PACF plot shows no values after k terms in AR(k) process.

Determining MA Terms

The ACF plot for the MA(k) process to show a strong correlation up to lag k and the decline sharply to indicate no correlation while the PACF plot is strong at lag k and trails onwards from k.

After preprocessing the time series model to make it stationary, any time series can be modeled in the following form: Forecast for y = constant + weighted sum of the last p values of y + weighted sum of the last q forecast errors

$$\hat{y} = \mu + \phi_1 yt - 1 + \ldots + \phi_p y_{t-p} + \theta_1 e_{t-1} + \ldots + \theta_q e_{t-q} \tag{5.1}$$

where, μ is a constant, ϕ_p is the AR coefficient at lag p, θ_q is the MA coefficient at lag q, and $e_{t-q} = \hat{y}_{t-q} y_{t-q}$ is the forecast error that was made at period $t - k$.

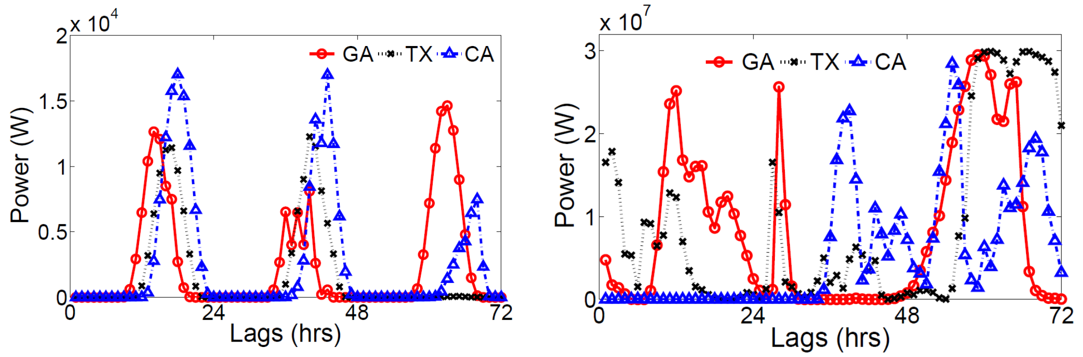


Figure 5.1: Hourly Solar and Wind Power Generated in Geo-Distributed Locations.

5.2 Prediction in Geo-Distributed MCS

Solar energy, workload, and electricity price traces were used in the design of an algorithm for scheduling workloads in geo-distributed data centers. Though the data vary at different locations, the data samples from the individual sensor exhibit the pattern.

Time series prediction techniques capture the daily and weakly seasonal pattern in the workload prediction and solar energy prediction using autoregressive and moving average models have given an accuracy of 95% to 93% for different workload traces such as WorldCup and NASA for prediction window of 1 hr. For solar energy trace for California, Texas, and Illinois Andreas and Stoffel (2017), the pattern is captured by SARIMA. Trace for February was used to learn the model. The model is fitted by changing the parameters to minimize the error observed in the residual plots and the ACF and PACF (Auto Correlation and Partial Auto Correlation) plots. The model is tested for the month of March. The error for different lags is almost constant starting with 23% for TX, and up to 25% for GA.

The varying nature of wind energy depends on different factors, such as temperature, pressure, wind directions. Wind energy is modeled using a moving average model. However the accuracy of prediction is 30% for a one-hour prediction window, and it reduces as the prediction window increases.

5.3 Prediction of Mobile Device/Users Contexts

Context such as location, WiFi state and battery level of the users is related to the behavior of the user. Many of these contexts repeat with daily or weekly patterns. The prediction scheme on the mobile device explore the correlation of the data and find patterns in the usage.

Location Prediction:

We design a new model for predicting the next hour location based on the following assumptions: 1. user's daily, weekly and hourly schedule creates a pattern of location changes. 2. common locations are derived from the connected Wi-Fi, data networks and GPS locations. History of the network and GPS location data is used to derive a model. The location prediction problem is stated as follows: Given a history of location data L^{hist}, predict location for next hour, given by $L^{\varrho}(t)$.

Using the history of a location, a transition matrix of location is developed. Matrix[i,j] each value shows the probability of transition from location[i] in previous time step to the current location[j]. This matrix is denoted by ν_{hr}. In order to capture the hourly usage patterns, we maintain a transition matrix for each hour of the day e.g. $\nu_0 \dots \nu_{23}$. We can also develop a pattern for each day of the week, or maintain transition matrices for weekend and weekdays separately.

In order to predict a location value for a given hour, ν_{hr} and last known location value is used to determine the most probable location $L^{\varrho}(t)$. .

$$L^{\varrho}(t) = g(\nu, L^{hist}). \tag{5.2}$$

With this prediction model (refer Figure.5.2), for a data of 10 days for a given user, we achieved an accuracy of 91.6%.

Device Usage Prediction: Device context prediction is the prediction of resource availability on the device. The resources predicted by this module are battery level and the type of network connectivity. We use a similar transition matrix for the battery and network connectivity prediction.

Battery State Prediction: The history of the battery state for a given device is obtained every 10 minutes. Markov state model shows the transition probability within different states that are based on the battery level as listed in Table 5.1. Similar to location transition matrix, hourly and daily transition models ϱ^{hr} and ϱ^{day} are used.

$$Soc(t) = G(\varrho^{hr}, \varrho^{day}) \tag{5.3}$$

Table 5.1: State v.s. Soc levels.

Battery State	**Soc range**
$S1$	$Soc > 0 \& Soc \leq 20$
$S2$	$Soc > 20 \& Soc \leq 40$
$S3$	$Soc > 40 \& Soc \leq 60$
$S4$	$Soc > 60 \& Soc \leq 80$
$S5$	$Soc > 80 \& Soc \leq 100$

Table 5.2: States of Network Connectivity.

State	**Network State**
β_0	WiFi
β_1	3G
β_2	No Connectivity

Figure 5.3 shows the battery prediction for next hour battery state for a single device. For 3 users, accuracy observed for predicted battery state is 80.8%.

Network State Prediction: Network connectivity data is obtained every 10 minutes. We consider three states of network connectivity, WiFi, 3G and No Connection as shown in Table 5.2. History of network connectivity is used to model hourly and daily patterns. $\beta(t) = G(\beta)$ where $\beta = (\beta^{hr}, \beta^{day})$ and G changes according to

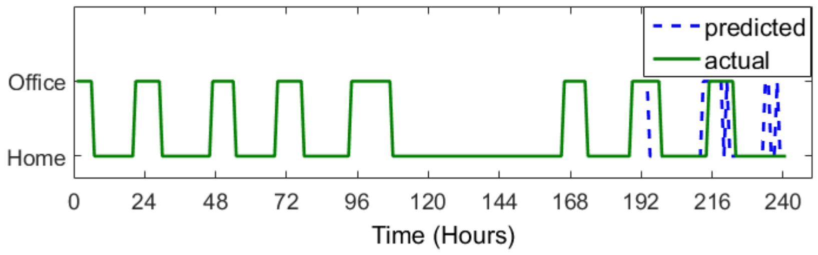


Figure 5.2: Hourly Location Prediction Results.

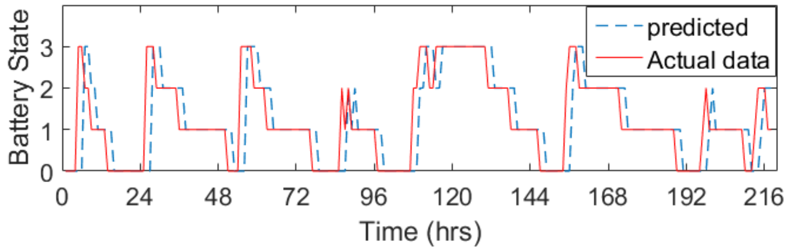


Figure 5.3: Predicted and Actual Battery SOC (Hourly).

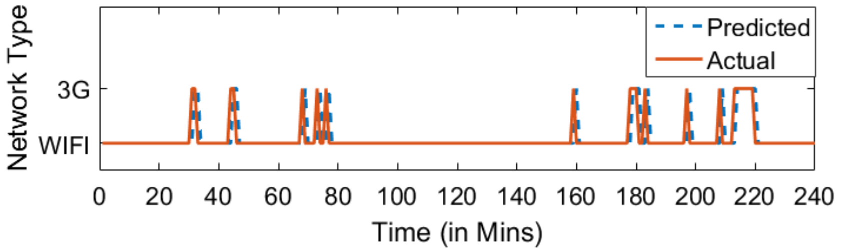


Figure 5.4: Predicted and Actual Network Type (Every 10 Mins).

changing pattern of the user.

With this prediction model (refer Figure.5.4), for a data of 10 days for 3 users, we observed the accuracy of 93.73%.

Chapter 6

OFFLOAD TO HETEROGENEOUS PLATFORMS

Many MCS applications use the monitoring of physiological signals or sensor data. In some edge applications, the data from the sensor is processed such that the results are given as feedback to the human user, or actuators. Hence the processing needs to complete within real-time constraints. Optimization discussed in Chapter 4 uses the application profile to meet the performance and energy requirements. This chapter characterizes the interference model, which may be used when different types of application run on a data center server.

Section 6.1 proposes an interference model for the colocation of an application within a server. The model is used to design a scheme to improve the energy and performance of data center servers. Section 6.2 mapping scheme of applications on many-core platforms. Example of Medical control applications that monitor vital signals of the patient and process it within time constraints.

6.1 Energy Aware Colocation in Data Center

The consolidation of applications increases the energy proportionality, where power consumption scales with the workload, and the idle power is almost zero. Many data center research works attempt to achieve such behavior using workload consolidation. However, consolidation causes degradation in performance due to contention in the shared resources, e.g., on-chip caches, buses, main memory, CPUs, and network. This research focuses on characterizing this interference in the colocated applications for improving the energy and performance of the offloaded application.

6.1.1 Interference Aware Colocation Policy

The interference effect of collocations on applications' performance has been studied using empirical methods in some recent works Mars *et al.* (2011b,a). The "Bubble-up" methodology predicts performance degradation due to the colocation of applications Mars *et al.* (2011b). It is used to design a colocation policy for delay-sensitive Google workloads. Authors inMars *et al.* (2011a) consider the applications' workload type as well as the effect of the underlying hardware to avoid performance degradation caused by collocation. This work addresses energy-efficient colocation of applications since the performance degradation (e.g., increasing the execution time of applications) is tightly correlated to the energy consumption.

When applications are run separately, they consume idle energy as well as the energy that depends on the utilization of the server. Energy consumption that the colocation incurs is estimated as a sum of utilization energy of individual applications, the idle energy as well as extra energy consumption due to interference. The approach is to develop a model to capture interference energy in colocated applications and use it to design application-aware colocation policy.

Interference Model

$$e^c = e^{idle} + e_i + e_k + e_{ik}^{ief}, \tag{6.1}$$

where e_i and e_k denotes the utilization energy consumption of tasks i and k respectively when running on the same system as standalone, e^{idle} denotes the idle energy to run either of the tasks i and k when running standalone, and e_{ik}^{ief} denotes additional energy consumption due to the colocation interference between tasks i and k which depends on the workload type of the tasks.

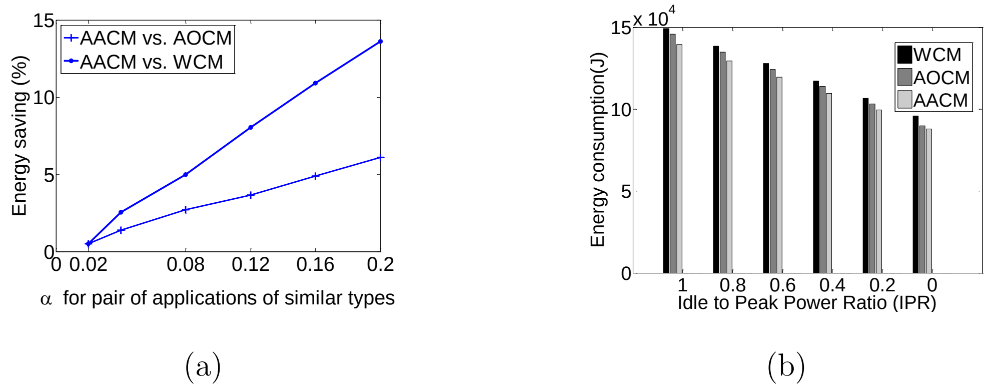


Figure 6.1: Energy Savings of AACM where **(a)** $\alpha = 0.02$ for Applications of Different Types, and **(b)** $\alpha = 0.02$ for Applications of Similar Types, And $\alpha = 0.1$ for Applications of Different Types over IPR Values.

6.1.2 Energy Aware Colocation Scheme for Server Applications

The models in Eq. 6.1 called the interference model give the interference coefficient obtained by running a set of applications in a server. This coefficient is used in a simulation-based evaluation to design a scheduling policy for an energy-aware colocation scheme.

The energy-aware colocation is formulated to minimize the total energy consumption of collocated sets such that all tasks are serviced and their performance (delay) requirement is met considering the interference energy. **AACM energy efficiency w.r.t. interference effect** Results in Figure 6.1(a) indicates that the energy saving of Application-Aware Colocation Management (AACM) linearly increases with increasing interference coefficient. **AACM energy efficiency w.r.t. IPR** Figure 6.1(b) shows that AACM saves energy over IPR (idle to peak ratio) value spectrum (i.e., 2-4.2% energy saving w.r.t. AOCM and 6-8% energy saving w.r.t. WCM).) since both idle power and utilization power contribute to interference energy and consequently in an energy saving of AACM.

6.2 Energy Aware Colocation on Many-Core Platforms

Some of the mobile applications are time-critical and have a low latency or throughput requirements that need to be addressed. The computational task involved is such that if it ran locally on the mobile device, the time constraints cannot be met. Hence such applications need edge processing. Medical Control Applications, real-time automation applications, real-time monitoring, video game, or perpetrator tracking are some of the example applications. Such applications leverage a local cloudlet or MEC server processing to achieve the latency or throughput requirements. Further, to achieve the desired throughput, the compute resources can involve highly efficient parallel processing architectures of many-core devices. In the case of continuous monitoring or complete automation, the data center application incurs a lot of energy consumption. This section explores the mapping of high throughput applications on different types of many-core devices to minimize the energy print.

6.2.1 Architecture Parallelism

High-performance computing architecture is used to meet the throughput requirements of the Edge Applications. In this work, the state of the art many-core devices is used. These are GPU devices, Intel Xeon Phi and Intel Core i7. While GPU and Intel Xeon Phi are externally attached and each has 1500 cores and 254 cores respectively. Intel Core i7, on the other hand, is a host processor and has 8 cores. Details of each of the architectures are provided in Table B.1. The number of cores and memory of each core determines the parallel threads that execute on the device and size of data that can be computed within each core.

6.2.2 Application Parallelism

The edge applications that execute on parallel architecture vary in the following ways:

1. Thread Parallelism: Each application can be composed of a code section that can utilize the threads available on the device. Some code sections can be a single thread in which case the remaining cores on the device are underutilized.

2. Data Level Parallelism: This involves executing the same computation on all the data points.

3. Processing Time and Energy: Mapping the application on different architecture result incur different performance such as throughput, time and energy.

4. In addition to the execution of the code, for GPU and Intel Xeon Phi, the time is required communication of data with the host device.

In this case, medical control applications are used. These applications have a requirement of high throughput. The results of the model predictive controller in each application need to be available within a certain time constraint to adjust the level of drug-infused to the patient. The characteristics of these edge applications used in the evaluation are a.) The Spatiotemporal model application, which is highly parallel application and results of every iteration are input to the next iteration. b.) Pharmacokinetic Model application is a serial application with limited thread and data-level parallelism. The code is optimized to execute the application on many-core devices.

6.2.3 Energy-Aware Mapping on Many-Core Devices

The evaluation includes profiling the application for throughput and energy on all three parallel computing platforms. The environment is set up such that multiple applications are running in parallel on the same many-core. Also, the applications are profiled to execute in combination. Based on the profile, the combination of application tasks that meet the throughput requirement and incur the lowest energy is observed to be: MIC executing the spatiotemporal model applications; i7 executing the pharmacokinetic model application. Details of parallelism of application, mapping on many-core devices and profiling are available in Appendix B

Chapter 7

POWER AWARE ELASTIC VM MANAGEMENT

MEC Base Station host multiple VMs to service the applications that are used by mobile users. As the MEC base station is near the edge of the network, multiple VMs are hosted to support the peak workloads of mobile users. This chapter proposes an elastic VM management system that scales the VMs up and down according to workload changes to save power for MEC base stations. The time-varying nature of the cloud applications' workload in LTE Base Station applications, the workload increases during the peak hours of the day and reduces during the night. The proactive VM management scheme is designed based on the prediction of the user pattern to scale the VMs according to the predicted workload. It also consolidates the VMs by addressing the overload and underload conditions. The evaluation consists of an OpenStack Cluster running application with real-world Internet traces on Intel travel-box clusters. The results indicate accuracy (15% prediction error) for the prediction of resources in the cluster.

7.1 Overview of Heuristic Algorithm

This algorithm executes in discrete time slots such that slot duration is small enough to capture the workload variabilities, yet long enough to predict the workload with reasonable accuracy and to avoid VM migration overhead. In each slot, the boundaries of VM resources are determined based on the prediction of VM work-load, and estimated resources. These boundaries are used to detect the overload and underload conditions in each host. The controller then schedules migration of the VM, optimizing for different goals on VM management such as minimizing migration costs, service interruption, power savings, etc.

7.2 Evaluation of VM Management Algorithm on OpenStack

OpenStack platform is Open Source software that enables infrastructure-as-a-service. It enables the management of virtual resources to build public or private clouds. The virtualized resources include compute, network, storage, identity, and images.

The OpenStack extension module is designed in two parts, resource monitoring, that collects the resource parameters such as CPU utilization of the host VMs, and dynamic VM management algorithm, which decides on the VM placement policies in the controller. Nova, OpenStack scheduler, initiates VM migration according to the decision of the VM management algorithm. Details of this algorithm are presented in Appendix C. A host monitors the resources of each VM and learns the seasonal pattern. The overload and underload conditions of each host are determined using estimated resource values. This information is evaluated in the scheduler which decides VM migration. This algorithm is evaluated on Intel OpenStack Cluster which services internet workload generated according to patterns of real-world traces. This algorithm estimates the resources with an accuracy of 85% and resulted in power savings of 23%.

Chapter 8

ContextAiDe EVALUATION

ContextAiDe architecture is evaluated using a Perpetrator Tracking Application. This perpetrator tracking application obtains video and image data from nearby user devices as the perpetrator moves from one location to another. The first part of this evaluation involves running the application on a set of participating devices, which includes mobile devices, fog server, and a cloud server available in the Impact lab at ASU. The second part demonstrates the use of ContextAiDe in the form of simulation on mobile data traces obtained by monitoring the participants from Yonsei University (LifeMap data set). This LifeMap dataset consists of timestamped data from different sensors on the user's mobile devices.

8.1 Perpetrator Tracking Application

All the mobile devices have ContextAiDe App installed. Perpetrator tracking app tasks run on the edge devices. The leader device initiates the task requests on the ContextAiDe publish-subscribe channel. Image capture and face detection are sensing and processing tasks. The user devices execute the image capture task while fog and cloud servers execute the face recognition tasks.

This application is useful to track a person by data mining on nearby mobile devices. The sensing task acquires media such as video or images from multiple users. The perpetrator app can help the authorities in searching for the perpetrator. Devices are selected to execute the MCS task based on the availability of context. Context requirements of the perpetrator tracking app are described in Table 8.1. It shows the exact and preferred contexts. User devices send the face data images to

Table 8.1: Context Requirements of Perpetrator Tracking App

Context	Value	Acceptable Deviation	Type	Details
Location	Event Location	0	exact	GPS value specify event location
Camera	Availability	0	exact	Ensures that the camera is available for use by ContextAiDe
Wifi	Availability	0	exact	Ensures connectivity to communicate the MCS data
Accelerometer Variance (m/s^2)	0	2	preferred	Reduces movement noise in camera output
location radius(m)	0	20	preferred	defines the search area for locating perpetrator
Orientation	300 °	20°	preferred	Ensures specific direction of viewing as requested by the investigator.

a fog or cloud server for face recognition. These results are used to estimate the perpetrator movement and predict the location in the next time step.

User Device Tasks

Image capture and face detection execute on the user device. *Image Capture* task starts the camera on the selected user device and captures images at the specified rate. *Face Detection* is called once every minute to process all the images captured in the previous time step.

Server Tasks

Face recognition executes on the Fog or Cloud server. Face data from mobile users is processed to recognize the face of the perpetrator. The face recognition app is designed using an open-source, python-based, face recognition library Ageitgey (2017) using a deep learning model. This model determines if any of the devices successfully captured the perpetrator's image.

8.2 Developer's and Volunteer's perspective

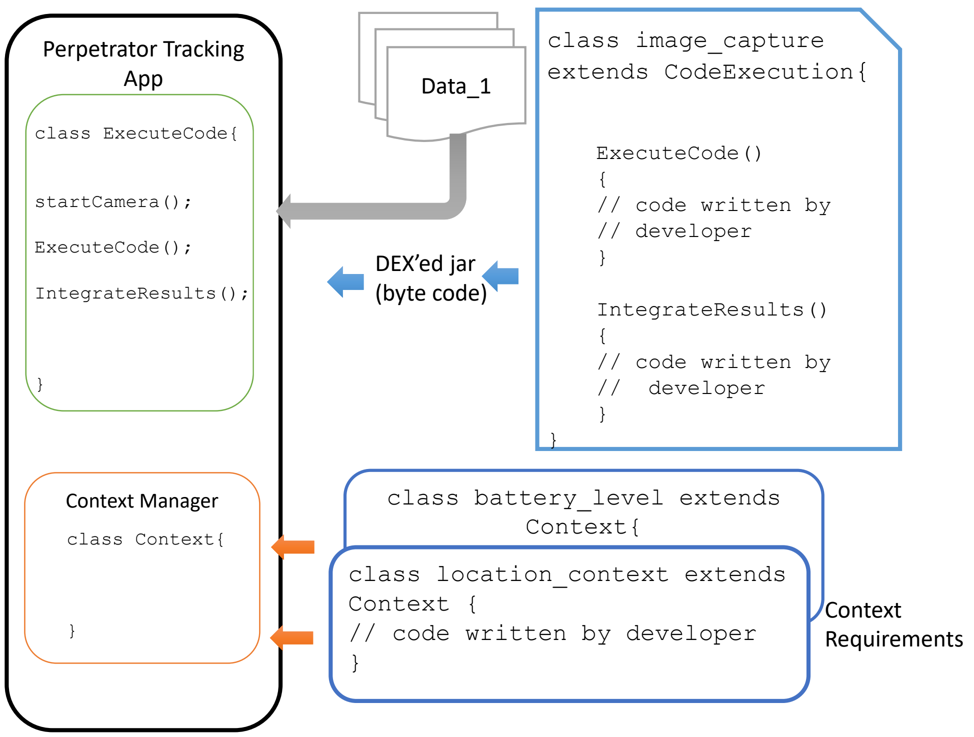
Developer's Perspective: ContextAiDe is designed to ease the developer from the burden of offload management of the application tasks. Developer integrates the MCS app by using the predefined API classes, e.g., the developer extends the Context class. The location context is defined using API is also shown in Code Listing 8.1. The listing shows the location context defined as the exact context as well as the preferred context. The distance function is also defined. The developer can build a class for any new sensor plugged into the device.

For example, if the developer wants to build a smart home application to optimize energy usage in the home, he can build contexts to trace the movement of different persons by including one or more techniques of indoor location sensing such as WiFi Triangulation, Bluetooth beacons Nath *et al.* (2018); Perera *et al.* (2018). These contexts can be used to further design optimization objectives such as controlled heating, and lighting by monitoring the location context of each person, and their movement in different rooms. The optimization algorithm in user recruitment needs to be defined to support new objectives, e.g., minimizing energy consumption. Additionally, the developer can also implement predictive schemes for the different contexts by monitoring user behavior or schedules of each person. In this case, the developer with use context prediction algorithms or design his own.

Listing 8.1: Definition of Context

```
public class location_context extends Context{
String context_name;
String context_type;   //exact or preferred
double weight ;        //[0-1]
datatype [] context_value;
float distance;
}
float[] location_val= new Float{13.065536,40.022937};
```

```
Context location_cntxt_z = new Context( "location", "exact",location_val, 0);
Context location_cntxt_p = new Context("location", "preferred", 0.7, 0.5);
```



Perpetrator Tracking App
class ExecuteCode{
startCamera();
ExecuteCode();
IntegrateResults();
}
Data_1
DEX'ed jar (byte code)
class image_capture extends CodeExecution{
ExecuteCode() { // code written by // developer }
IntegrateResults() { // code written by // developer }
}
Context Manager
class Context{ }
class battery_level extends Context{
class location_context extends Context { // code written by developer }
Context Requirements

Figure 8.1: Execution Mechanism using DEX for Android.

Further, the architecture provides the developer with automatic offloading of the application tasks across the pool of devices with predefined classes. Figure 8.1 shows offload management and execution. OffloadManagement class is used to define the offloaded task of the application and the application logic. ExecuteCode() method of this class defines the application logic, and IntegrateResults() method defines logic for processing the results from multiple devices. The offloaded task, in the form of DEX code, is sent on the android user device. ExecuteCode() method invokes the execution of the task on the user device.

Volunteer's Perspective: The volunteer configures the App to set limits on the resources to be used by the offloaded tasks. A new offload task is accepted if its estimated resource usage is within limits, thus protecting the resources for the owner's

Table 8.2: Perpetrator Location Tracking Results

Plot(XS,CA)	**Value**	**Description**
(a) tracked	Location	plot indicates perpetrator tracked within search area with both strategies
(b) 542,221	Radius	plot indicates search radius (here mean).
(c) 86%,31.7%	Devices	Number of devices (here percentage)
(d) 7.5MB,2.8MB	Data	Average data sent by sensing devices to fog server every minute.
(e) 0.459,0.335	CSI	lower CSI indicates better Contextual data acquired.
(f) 59.6J, 22.6J	Energy	Average Mobile Energy Consumption every minute.
(g) 17.9s, 14.8s	Time	Average time incurred for processing a request.

use.

8.3 Evaluation Run

This experimental evaluation is done using 14 mobile devices, one notebook, a fog server, and a cloud server. Specification details of these devices are listed as follows:

1. 6 One Plus One phones, Qualcomm Snapdragon 2.5GHz Quad core CPU, 3GB RAM, 64GB Storage, Android 5.1.1
2. 4 Nexus 5 phone, Qualcomm Snapdragon 2.3GHz Quad core CPU, 2 GB RAM, 32GB Storage, Android 5.1.1
3. 2 LG g2 phone, Qualcomm Snapdragon 2.26GHz Quad core CPU, 2 GB RAM, 16GB Storage, Android 5.1.1
4. 1 Nexus 7 tablet, Qualcomm Snapdragon 1.5GHz CPU, 2 GB RAM, 32GB Storage, Android 5.1.1
5. 1 Moto G5 plus phone, Qualcomm Snapdragon 2.0GHz CPU, 4 GB RAM, 64GB Storage, Android 7.0
6. Fog Server: Intel i7 Desktop 3.5GHZ Quad core CPU, 16GB RAM
7. Cloud server in GoogleCloud with 8vCPUs and 32 GB memory

A perpetrator tracking app is used to evaluate ContextAiDe. It tracks the perpetrator, as he moves through a series of locations. This evaluation is performed on14 android devices with varying context values. The search begins from the last seen location of the perpetrator and a fix search radius. At a given location, ContextAiDe optimally selects user devices to acquire images from a camera. MCS app components have two processing components, 1. detecting images that contain faces (runs on a mobile device which captures images.) 2. recognize the face in the images sent (runs on the Fog server).

The application run is of 30 minutes. There are14 mobile devices, and one notebook participating in the evaluation. One of the mobile devices acts as leader devices. The leader device discovers the devices available nearby and recruits some of the devices to execute the MCS task. This evaluation compares ContextAiDe to the existing user recruitment strategy (XS). The existing strategy uses the exact context of location and WiFi availability for optimal selection of users in a fixed search radius. ContextAiDe strategy involves the optimal selection of devices to adapt to changing context changes and requirements of the application. Table 8.2 shows the performance of ContextAiDe vs. XS in a single request which tracks perpetrator through location 1 to the 10.

This Figure reffig:loctrac shows the evaluation of ContextAiDe and XS user recruitment. The contexts on the user devices change during the execution. Initial the Contextual requirements of the Perpetrator tracking app are user orientation of 240°, search radius of 10m and images rate of 6 images/min.

8.3.1 *ContextAiDe Evaluation with Other Recruitment Strategies*

This evaluation is designed using LifeMap Dataset Chon *et al.* (2012). 10 Perpetrator tracking app requests that last for an average duration of 30 mins are gen-

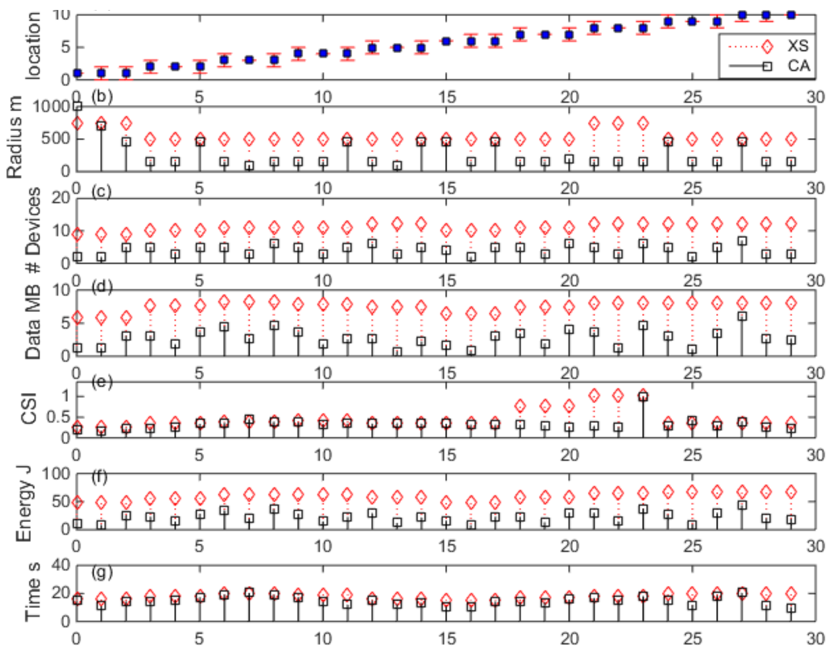


Figure 8.2: Perpetrator Tracking Scenario.

erated. In each request, the perpetrator is tracked as he moves through the campus area. Evaluation of ContextAiDe user recruitment is shown compared to prior research works. Different aspects of user recruitment strategy such as accuracy, energy savings, data transfer costs are analyzed.

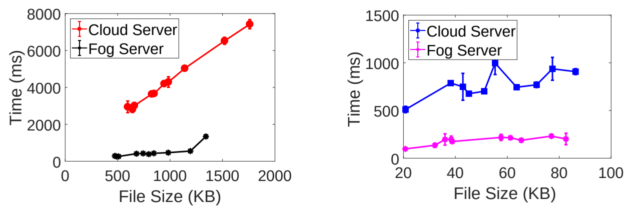
LifeMap Dataset: Real-life mobile data traces from obtained from free moving users in Yonsei University are used Chon *et al.* (2012). These traces consist of users monitored using multiple sensors over a few months. The trace includes timestamped data of their GPS location, WiFi and data connectivity, user activity, and battery level. The data granularity is 2 mins.

8.3.2 Profiling Execution and Communication Time

Profiling File Transfer Time One of the common methods to upload camera images to the server is saving the file locally on the device and sending it to the server using multipart file upload. Figure 8.3a shows the time incurred to send the original camera images to Cloud as well as the Fog server. Different file sizes are obtained by using multiple resolution settings of Camera to capture images ranging from 13M to 1M which are sent using lossless compression. Time is noted on the Android device when it starts to send data, up to the time the acknowledgment is received from the server. Another approach is to stream the data from the camera to the cloud. Figure 8.3b shows the time incurred to stream the image data from the camera to the server. Streaming methods allow capturing camera data in preview mode which are smaller files compared to original images that can be saved on the device. In both methods, at the server end, the PHP code stores image data into a file before it is used by face recognition application.

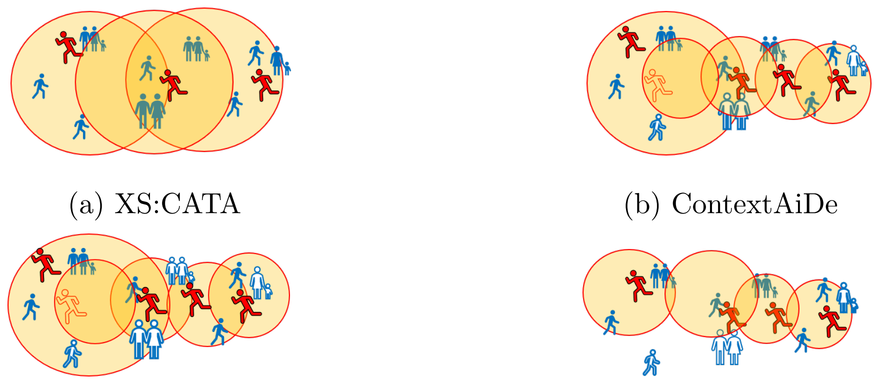
Profiling Data Sensing Time The time for sensing is noted on the Android device from the start of sensing requests received on the device to the time data is available on the device for transfer. In the perpetrator tracking application, the data sensing time is noted for the acquisition of image files and the data stream from the camera. Capturing images of different file sizes, the sensing time was observed to be 2203.9 ms with a standard deviation of 397 ms.

Profiling Execution Time of MCS Tasks Execution of the perpetrator tracking MCS application involves running of face detection application on the mobile device and face recognition application on the Fog/Cloud server. The time for running the face detection task is noted from the start of data available until the results of face detection applications are available. The average time incurred for varying file sizes is 1161 ms with a stan-



(a) File Transfer (b) Stream Process

Figure 8.3: Comparison of File Transfer Time in Fog vs. Cloud



(a) XS:CATA (b) ContextAiDe

(c) ContextAiDe with Stochastic (d) Ideal

Figure 8.4: Comparison of User Recruitement Strategies.

dard deviation of 331.69 ms while the face recognition application task requires an average 860 ms with a standard deviation of 101.8 ms.

For evaluation run (§8.3) and the perpetrator tracking app (§8.1), the time is estimated starting from request for image capture to the time when results of face recognition are generated as a combined time for sensing and data transfer time for a single image. The time incurred using the file saving method is estimated to be 8.6s for cloud and 4.71s for Fog server processing (file saving method). The estimated time for stream processing is around 5s for cloud and 4.414s using the Fog server.

8.3.3 Strategies for User Recruitment

In this section, different user recruitment approaches are described in detail. Figure 8.4

1. Existing Strategy (**XS**): This user recruitment scheme is similar to Context-Aware Task Allocation (CATA) Hassani *et al.* (2015) where context optimized selection is performed for each new task request. The next location is given by one of the devices that recognize the perpetrator while the search radius is fixed as seen in Figure 8.4a. 2. ContextAiDe's Current Available Context (**CA**): This approach employs a user recruitment scheme discussed in the § 4.1.1. Currently available context on

the device is used by this approach. Optimization is performed for user recruitment when the required context becomes suddenly unavailable or context request is initiated. If the perpetrator being tracked is found, the search radius is based on context availability and projected movement. This can be seen in Figure 8.4b. In case the person is not found the search radius is increased.

3. ContextAide's Stochastic(**CAS**): This approach is similar to the CA approach but the context used in device selection is the predicted context of the device. Context is predicted using a stochastic approach. Figure 8.4c create focused search to obtain more relevant data. 4. Ideal (**I**): This recruitment revises the selection decision by performing optimization at fixed time intervals to select optimally context matching devices based on actual values each time. In the evaluation presented, the time interval is set for 2 mins. This recruitment scheme chooses devices with the best CSI in every time step to provide optimal user selection. Figure 8.4d shows correct selection devices knowing the future values of context on the devices. 5. ContextAiDe's Cloud(**CAC**): This uses user recruitment strategy (§ 4.1.1) but cloud is used for processing.

ContextAiDe's user recruitment schemes are designed to adapt to dynamically changing context requirements, e.g. location or search radius may change based on the movement of the perpetrator at runtime. New location and search radius are based on the previous evaluation.

8.3.4 Evaluation using Data Traces.

In this section, ContextAiDe is evaluated for strategies discussed in § 8.3.3. Parameters are as follows:

(1.) CSI: Lower CSI value indicates data acquired matches the required context.

(2.) Data Usage: This value indicates data transferred between the Edge devices

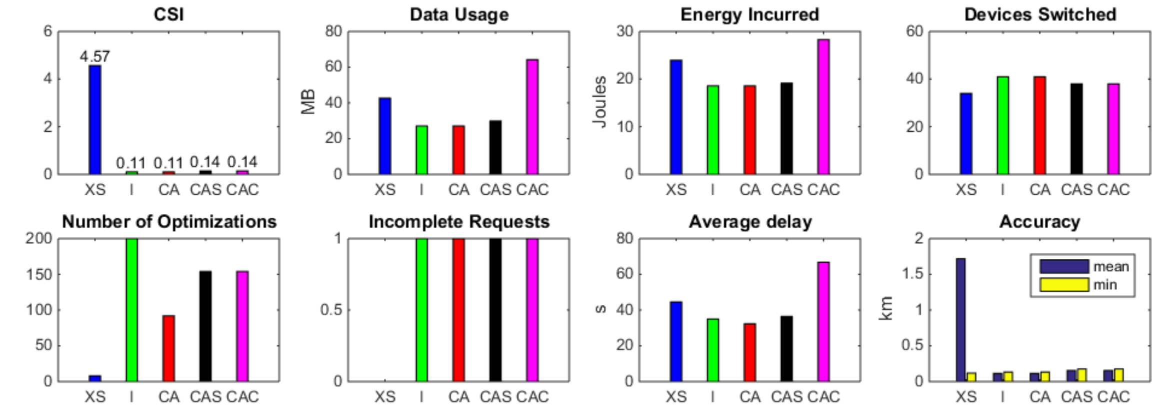


Figure 8.5: Performance of ContextAiDe w.r.t. Other Architectures.

when it is transferred to fog/cloud. The amount of data transferred by different algorithms is compared.

(3.) Energy: The energy is estimated based on data sensing, processing and data transfer that are executed on the mobile device (§ 8.3) while processing each application request.

(4.) Number of Optimizations: In a given sequence of application requests, optimization initiated depends on the user recruitment algorithm used. The number of optimizations indicates the operational overhead of the algorithm.

(5.) Devices switched: Sum of new devices that were recruited during execution.

(6.) Incomplete requests: Context-aware recruitment of devices causes some of the requests to be incomplete. A total number of request that remains incomplete over the sequence of requests.

(7.) Delay: Average time incurred to accomplish the MCS task.

(8.) Accuracy: It shows the distance between the actual and the estimated location of the perpetrator.

Figure 8.5 shows that CA successfully generates context-relevant data shown by lower CSI and accuracy plot. Proactive context estimation results in little better energy and delay performance for CAS. Data used in CAS is 24.8% lesser and energy is 37.8% lower than existing strategy (XS) for accurately tracking the perpetrator. Time incurred is improved by 43%. This is achieved using CSI optimization indicated by the accuracy plot. Close monitoring of context and the proactive decision based on expected context values results in savings of data usage energy consumption and time incurred. However, this comes at the cost of accuracy. The minimum distance achieved in CA, CAS is a little more than XS and I strategy but the mean value of XS is much higher than CA/CAS. Alternatively, using the cloud server in CAC strategy incurs much higher energy (33%) and delay (50%) than CAS.

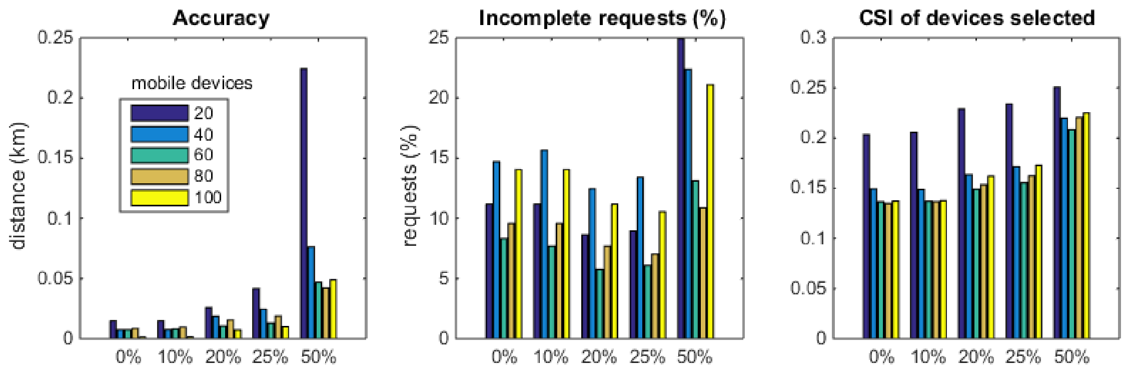


Figure 8.6: Performance of ContextAiDe in varying Uncertainty Scenarios.

Figure 8.5 shows that CA successfully generates context-relevant data shown by lower CSI and accuracy plot. CAS uses a proactive context and results in better energy and delay performance. Data used in CAS is 24.8% lesser, and energy is 37.8% lower than the existing strategy (XS) for accurately tracking the perpetrator. Time incurred is improved by 43%. This is achieved using CSI optimization indicated by the accuracy plot. Monitoring of context and the proactive decision based on expected context values results in savings of data usage, energy, and time incurred. However, this comes at the cost of accuracy. The minimum distance achieved in CA and CAS is a little more than XS and I strategy, but the mean value of XS is much higher than CA/CAS. Alternatively, using the cloud server in CAC strategy incurs much higher energy (33%) and delay (50%) than CAS. CAS strategy is evaluated using a large scale simulation set up. Variation of the number of mobile users is assumed for executing the Perpetrator tracking app. The mobility, WiFi, battery, and activity are modeled according to the LifeMap usage data. The performance of CAS is evaluated for variation in uncertainty that is associated with Contextual data. Figure 8.6 shows the variation in accuracy, percentage of incomplete request (due to unavailability of context), and average CSI, which shows context relevance of data obtained. For lower uncertainty ($< 20\%$), accuracy is as desired less than 40 m. Percentage of requests abandoned and CSI of devices selected remain steady for lower uncertainty ($< 25\%$). But as uncertainty increases, higher values of CSI are noted for high at the cost of increased incomplete requests and lower accuracy.

Chapter 9

CONCLUSION AND FUTURE DIRECTIONS

9.1 Discussion of Solutions

ContextAiDe Architecture enables the design of MCS apps, that meets the real-time requirements of the application. Many components of this architecture independently address various problems of execution of crowdsensing on Edge. This section summarizes various solutions presented in this thesis towards the design of ContextAiDe architecture.

1. **Context Optimized User Recruitment** MCS requirements are described in the form of Context and its values that are useful to execute the different MCS tasks, such as data acquisition, processing data, run analytics. Recruitment based on meeting the context requirements has two-fold advantages. The data acquired is relevant to the MCS tasks and selective data can be processed quickly using the edge devices. MCS application with real-time requirements such as perpetrator tracking needs video and image data from nearby mobile users. It needs data from users in a specific location and fast processing of analytical tasks to recognize the perpetrator. Chapter 3 and Chapter 4 provide details of context specification and context optimization-based user requirement. Additionally, to address the uncertainty associated with mobile usage, a stochastic optimization scheme is designed. Additionally, Chapter 4 describes various components of ContextAiDe architecture and their functionality. These components can be integrated into MCS app through API calls. Developers can build communication, execution, and context optimized user recruitment for

their MCS app using ContextAiDe. A detailed example of perpetrator tracking is explained in Chapter 8 and Pore *et al.* (2019).

2. **Human Device Interaction Models** The interaction of humans is captured on the user devices and mapped to various activities of the day. These contexts repeat with daily and/or weekly seasonality. Chapter 5 presents a novel algorithm that runs on user devices to characterize human behavior into models. These models are used to design proactive scheduling strategies and proactive user recruitment strategies in ContextAiDe applications. User recruitment with prior knowledge of context information benefits the execution of MCS with minimal interruptions.

3. **Power-aware Elastic VM Scheduling Algorithm** LTE Base Station host VMs which run services for mobile users. The workload in a given base station follows daily and weekly usage patterns. Chapter 7 and C presents details of the VM management algorithm which scales the VM according to workload thus consolidating VMs during less busy hours. This saves energy for the data centers.

4. **Offloaded applications on many-core devices** Some of the MCS applications have high computational requirements with real-time constraints. Many-core devices such as GPU, Intel Xeon Phi support highly parallel execution of applications. Many-core device architecture varies in the number of cores, computational power, thread availability, and the cache memory structure. The offloaded applications also vary in the thread and data-level parallelism. High throughput application, when mapped on many-core exhibits different performance, and energy print. This energy-aware mapping of applications on many-core architectures is used to save energy many-core servers. Chapter 6 explains

the energy-aware mapping scheme of applications on many-core devices.

5. **Colocated VM Scheduling** The efficient scheduling of colocated VMs and data center applications is based on the energy interference model. Colocated applications that execute on the same server utilize the same set of resources. This results in contention of resources which is modeled as the interference. An energy-aware mapping scheme based on this interference model reduces the energy consumption in the data center. Chapter 6 explains the energy-aware mapping scheme of colocated applications.

6. **Evaluation of ContextAiDe:** ContextAiDe architecture is evaluated on a small scale on Android devices, edge servers, and cloud by distributing the tasks of a perpetrator tracking app in the edge devices. Large-scale evaluation of ContextAiDe shows savings in energy, latency, and improvement in the quality of output of the executed app using simulation. It also evaluates the MCS application for successful tracking of the perpetrator in an uncertain user environment. All the results are presented in Chapter 8 and Pore *et al.* (2019).

9.1.1 Future Work

1. **ML Models of User Contexts:** ContextAiDe evaluation shows the advantage of using context prediction models in stochastic optimization for recruitement of MCS users. The user behavior and changes in mobile device contexts can be shared across multiple users. MEC based architecture can be designed to share the ML models, store, and use them for participating users. This prediction model can allow the inclusion of new users with better knowledge about their contexts.

2. **Anticipatory services for MEC:** Anticipatory design of services is gaining

huge attention currently Ntalampiras and Fiore (2018), specifically for MEC. The Edge applications are either streaming services such as video, live data streaming or MCS applications that acquire data from a large set of users and process it in real-time. Important requirements of such applications are low latency and high bandwidth. In edge application such as video streaming discussed by Yang et al., MEC implement rate adaptation and video caching for the mobile user. In such a situation predictive and machine learning models benefit the service quality of the MEC app. Human interaction models focused on mobility and human activity can be used extensively to design anticipatory tasks in MEC. Schemes for preemptive caching of data and stateful VM migrations with checkpoints to support mobility can be designed in future MEC architectures.

REFERENCES

Mike Wheatley, "Report: Simple economics will make edge computing vital for Internet of Things", https://siliconangle.com/2016/11/11/report-simple-economics-will-make-edge-computin online; accessed 12 Dec 2019 (2016).

Abbasi, Z., M. Pore, A. Banerjee and S. K. Gupta, "Multi-tier energy buffering management for idcs with heterogeneous energy storage devices", in "International Conference on High performance Computing Conference (HiPC2013)", (2013a).

Abbasi, Z., M. Pore and S. K. Gupta, "Impact of workload and renewable prediction on the value of geographical workload management.", in "Second International Workshop on Energy Efficient Data Centers (E2DC), held as a part of ACM eEnergy", (2013b).

Ageitgey, "Face Recognition", https://github.com/ageitgey/face_recognition, online; accessed 29 November 2017 (2017).

Andreas, A. and T. Stoffel, "NREL solar radiation research laboratory (SRRL).baseline measurement system (BMS); golden, colorado (data); NREL report no. da-5500-56488", Downloaded from http://www.nrel.gov/midc/ (2017).

Banerjee, A. and S. K. Gupta, "Analysis of smart mobile applications for healthcare under dynamic context changes", IEEE Trans. on Mobile Computing **14**, 5, 904–919 (2015).

Bao, X. and R. Roy Choudhury, "Movi: mobile phone based video highlights via collaborative sensing", in "Proc. of the 8th international conference on Mobile systems, applications, and services", pp. 357–370 (ACM, 2010).

Baresi, L., S. Guinea and D. F. Mendonca, "A3droid: A framework for developing distributed crowdsensing", in "2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)", pp. 1–6 (2016).

Bradai, S., S. Khemakhem and M. Jmaiel, "Re-opsec: Real time opportunistic scheduler framework for energy aware mobile crowdsensing", in "24th Intl. Conf. on Software, Telecommunications and Comput. Netw. (SoftCOM)", pp. 1–5 (2016).

Brouwers, N. and K. Langendoen, "Pogo, a middleware for mobile phone sensing", in "Proc. of the 13th Intl. Middleware Conf.", pp. 21–40 (Springer-Verlag New York, Inc., 2012).

Capponi, A., C. Fiandrino, D. Kliazovich, P. Bouvry and S. Giordano, "A cost-effective distributed framework for data collection in cloud-based mobile crowd sensing architectures", IEEE Transactions on Sustainable Computing **2**, 1, 3–16 (2017).

Cardone, G., L. Foschini, P. Bellavista, A. Corradi, C. Borcea, M. Talasila and R. Curtmola, “Fostering participaction in smart cities: a geo-social crowdsensing platform”, IEEE Communications Magazine **51**, 6, 112–119 (2013).

Chen, H., B. Guo, Z. Yu and Q. Han, “Toward real-time and cooperative mobile visual sensing and sharing”, in “IEEE INFOCOM 2016 - The 35th Annual IEEE Intl. Conf. on Computer Communications”, pp. 1–9 (2016).

Chon, Y., E. Talipov, H. Shin and H. Cha, “CRAWDAD dataset yonsei/lifemap (v. 2012-01-03)”, Downloaded from `https://crawdad.org/yonsei/lifemap/20120103` (2012).

Crane, L., “Right time, right place: A collaborative approach for accurate context-awareness in mobile apps and ads”, Downloaded from https://www.cs.columbia.edu/2015/collaborative-place-models/ (2015).

da Rosa, J. H., J. L. Barbosa and G. D. Ribeiro, “Oracon: An adaptive model for context prediction”, Expert Systems with Applications **45**, Supplement C, 56 – 70, URL `http://www.sciencedirect.com/science/article/pii/S0957417415006302` (2016).

Fernando, N., S. W. Loke and J. W. Rahayu, “Honeybee: A programming framework for mobile crowd computing”, in “MobiQuitous”, (2012).

Fiandrino, C., F. Anjomshoa, B. Kantarci, D. Kliazovich, P. Bouvry and J. N. Matthews, “Sociability-driven framework for data acquisition in mobile crowdsensing over fog computing platforms for smart cities”, Sustainable Computing, IEEE Transactions on **2**, 4, 345–358 (2017).

Garcia Lopez, P., A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber and E. Riviere, “Edge-centric computing: Vision and challenges”, SIGCOMM Comput. Commun. Rev. **45**, 5, 37–42, URL `http://doi.acm.org/10.1145/2831347.2831354` (2015).

Goh, C. Y., J. Dauwels, N. Mitrovic, M. T. Asif, A. Oran and P. Jaillet, “Online map-matching based on hidden markov model for real-time traffic sensing applications”, in “2012 15th Intl. IEEE Conf. on Intelligent Transportation Systems”, pp. 776–781 (2012).

Han, K., C. Zhang and J. Luo, “Taming the uncertainty: Budget limited robust crowdsensing through online learning”, IEEE/ACM Transactions on Networking **24**, 3, 1462–1475 (2016).

Hardawar, D., “Driving app waze builds its own siri for hands-free voice control”, (2012).

Hassani, A., P. D. Haghighi and P. P. Jayaraman, “Context-aware recruitment scheme for opportunistic mobile crowdsensing”, in “IEEE 21st Intl. Conf. on Parallel and Distributed Systems (ICPADS)”, pp. 266–273 (2015).

Heng Zhang, H. W. S. B. R. K. P., Nawanol Theera-Ampornpunt, "Sense-aid: A framework for enabling network as a service for participatory sensing", in "Middleware", (ACM, 2017).

Herrera, J. C., D. B. Work, R. Herring, X. J. Ban, Q. Jacobson and A. M. Bayen, "Evaluation of traffic data obtained via gps-enabled mobile phones: The mobile century field experiment", Transportation Research Part C: Emerging Technologies **18**, 4, 568–583 (2010).

Higashino, T., H. Yamaguchi, A. Hiromori, A. Uchiyama and K. Yasumoto, "Edge computing and iot based research for building safe smart cities resistant to disasters", in "2017 IEEE 37th Intl. Conf. on Dist. Comp. Systs. (ICDCS)", pp. 1729–1737 (2017).

Hu, S., L. Su, H. Liu, H. Wang and T. F. Abdelzaher, "Smartroad: Smartphone-based crowd sensing for traffic regulator detection and identification", ACM Trans. Sen. Netw. **11**, 4, 55:1–55:27, URL `http://doi.acm.org/10.1145/2770876` (2015).

Hull, B., V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan and S. Madden, "Cartel: a distributed mobile sensor computing system", in "Proc. of the 4th international conference on Embedded networked sensor systems", pp. 125–138 (ACM, 2006).

Jayaraman, P. P., J. B. Gomes, H.-L. Nguyen, Z. S. Abdallah, S. Krishnaswamy and A. Zaslavsky, "Scalable energy-efficient distributed data analytics for crowdsensing applications in mobile environments", IEEE Trans. on Computational Social Systems **2**, 3, 109–123 (2015).

Jayaraman, P. P., C. Perera, D. Georgakopoulos and A. Zaslavsky, "Efficient opportunistic sensing using mobile collaborative platform mosden", in "Collaboratecom", pp. 77–86 (IEEE, 2013).

Kenneth research, "Edge computing market - industry insights by growth, emerging trends and forecast by 2023", Downloaded from https://www.marketwatch.com/press-release/edge-computing-market—industry-insights-by-growth-emerging-trends-and-forecast-by-2023-2019-09-04 (2019).

Khan, M. A., H. Debnath, N. R. Paiker, N. Gehani, X. Ding, R. Curtmola and C. Borcea, "Moitree: A middleware for cloud-assisted mobile distributed apps", in "4th Intl. Conf. on MobileCloud", pp. 21–30 (IEEE, 2016).

Kotz, D., T. Henderson, I. Abyzov and J. Yeo, "CRAWDAD dataset dartmouth/campus (v. 2009-09-09)", Downloaded from http://crawdad.org/dartmouth/campus/20090909 (2009).

Lee, K. and I. Shin, "User mobility-aware decision making for mobile computation offloading", in "IEEE 1st Intl. Conf. on CPSNA,", pp. 116–119 (IEEE, 2013).

Lv, Q., Y. Qiao, N. Ansari, J. Liu and J. Yang, "Big data driven hidden markov model based individual mobility prediction at points of interest", IEEE Trans. on Vehicular Technology **66**, 6, 5204–5216 (2017).

Mars, J., L. Tang and R. Hundt, "Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity", IEEE Comput. Archit. Lett. **10**, 2, 29–32, URL `http://dx.doi.org/10.1109/L-CA.2011.14` (2011a).

Mars, J., L. Tang, R. Hundt, K. Skadron and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations", in "Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture", pp. 248–259 (ACM, 2011b).

Meurisch, C., K. Planz, D. Schäfer and I. Schweizer, "Noisemap: Discussing scalability in participatory sensing", in "Proc. of 1st Intl. Workshop on SENSEMINE", SENSEMINE'13, pp. 6:1–6:6 (2013).

Miluzzo, E., N. D. Lane, S. B. Eisenman and A. T. Campbell, "Cenceme: Injecting sensing presence into social networking applications", in "Proc. of 2Nd Euro. Conf. on Smart Sensing and Context", pp. 1–28 (2007).

Mokryn, O., D. Karmi, A. Elkayam and T. Teller, "Help me: Opportunistic smart rescue application and system", in "The 11th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net), 2012", pp. 98–105 (IEEE, 2012).

Nath, R. K., R. Bajpai and H. Thapliyal, "Iot based indoor location detection system for smart home environment", in "2018 IEEE International Conference on Consumer Electronics (ICCE)", pp. 1–3 (IEEE, 2018).

Ntalampiras, S. and M. Fiore, "Forecasting mobile service demands for anticipatory mec", in "2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)", pp. 14–19 (2018).

Oskooyee, K. S., A. Banerjee and S. K. Gupta, "Neuro movie theatre: A real-time internet-of-people based mobile application", in "The 16th Intl. Workshop on Mobile Computing Systems and Applications", (2015).

Oskooyee, K. S., A. Banerjee, J. Sohankar and S. K. S. Gupta, "Safedrive: An autonomous driver safety application in aware cities", in "IEEE PerCom Workshops", pp. 1–6 (2016).

Panichpapiboon, S. and P. Leakkaw, "Traffic density estimation: A mobile sensing approach", IEEE Communications Magazine **55**, 12, 126–131 (2017).

Perera, C., S. Aghaee, R. Faragher, R. Harle and A. F. Blackwell, "Contextual location in the home using bluetooth beacons", IEEE Systems Journal (2018).

Pore, M., Z. Abbasi, S. K. Gupta and G. Varsamopoulos, "Energy aware colocation of workload in data centers", in "19th International Conference on High Performance Computing (HiPC), 2012", pp. 1–6 (IEEE, 2012).

Pore, M., A. Banerjee and S. Gupta, "Performance evaluation of multi core systems for high throughput medical applications involving model predictive control", in "High Performance Computing (HiPC), 2014 21st International Conference on", pp. 1–10 (2014).

Pore, M., A. Banerjee, S. Gupta and H. K. Tadepalli, "Performance trends of multi-core system for throughput computing in medical application", in "2nd Workshop on Performance Engineering and Applications at HIPC", (2013).

Pore, M., V. Chakati, A. Banerjee and S. K. S. Gupta, "Contextaide: End-to-end architecture for mobile crowd-sensing applications", ACM Trans. Internet Technol. **19**, 2, 19:1–19:23, URL `http://doi.acm.org/10.1145/3301444` (2019).

Pore, M., K. S. Oskooyee, V. Chakati, A. Banerjee and S. K. Gupta, "Enabling real-time collaborative brain-mobile interactive applications on volunteer mobile devices", in "Proceedings of the 2nd International Workshop on Hot Topics in Wireless", pp. 46–50 (ACM, 2015).

Ra, M.-R., B. Liu, T. F. La Porta and R. Govindan, "Medusa: A programming framework for crowd-sensing applications", in "Proc. of the 10th Intl. Conf. on MobiSys", pp. 337–350 (2012).

Reddy, S., D. Estrin and M. Srivastava, "Recruitment framework for participatory sensing data collections", in "Intl. Conf. on Pervasive Computing", pp. 138–155 (Springer, 2010).

Sapienza, M., E. Guardo, M. Cavallo, G. L. Torre, G. Leombruno and O. Tomarchio, "Solving critical events through mobile edge computing: An approach for smart cities", in "2016 IEEE Intl. Conf. on Smart Comp.", pp. 1–5 (2016).

Satyanarayanan, M., "Mobile computing: the next decade", in "Proceedings of the 1st ACM workshop on mobile cloud computing & services: social networks and beyond", p. 5 (ACM, 2010).

Sherchan, W., P. P. Jayaraman, S. Krishnaswamy, A. Zaslavsky, S. Loke and A. Sinha, "Using on-the-move mining for mobile crowdsensing", in "13th Intl. Conf. on Mobile Data Management (MDM)", pp. 115–124 (IEEE, 2012).

Shi, J. and W. Jia, "Real-time target tracking through mobile crowdsensing", in "Web Information Systems Engineering – WISE 2017", edited by A. Bouguettaya, Y. Gao, A. Klimenko, L. Chen, X. Zhang, F. Dzerzhinskiy, W. Jia, S. V. Klimenko and Q. Li, pp. 3–18 (Springer International Publishing, Cham, 2017).

Simoens, P., Y. Xiao, P. Pillai, Z. Chen, K. Ha and M. Satyanarayanan, "Scalable crowd-sourcing of video from mobile devices", in "Proc. of the 11th Annual Intl. Conf. on MobiSys", pp. 139–152 (ACM, 2013).

Sun, J., X. Zhu, C. Zhang and Y. Fang, "Rescueme: Location-based secure and dependable vanets for disaster rescue", IEEE Journal on Selected Areas in Communications **29**, 3, 659–669 (2011).

Toledano, E., D. Sawada, A. Lippman, H. Holtzman and F. Casalegno, "Cocam: A collaborative content sharing framework based on opportunistic p2p networking", in "CCNC", pp. 158–163 (IEEE, 2013).

Tran, T. X., A. Hajisami, P. Pandey and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges", IEEE Commun. Mag. **55**, 4, 54–61 (2017).

Wang, L., D. Zhang, D. Yang, A. Pathak, C. Chen, X. Han, H. Xiong and Y. Wang, "Space-ta: Cost-effective task allocation exploiting intradata and interdata correlations in sparse crowdsensing", ACM Trans. Intell. Syst. Technol. **9**, 2, 20:1–20:28, URL `http://doi.acm.org.ezproxy1.lib.asu.edu/10.1145/3131671` (2017a).

Wang, S., R. Urgaonkar, T. He, K. Chan, M. Zafer and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs", IEEE Trans. Parallel Distrib. Syst. **28**, 4, 1002–1016 (2017b).

Wu, Y., Y. Wang, W. Hu and G. Cao, "Smartphoto: A resource-aware crowdsourcing approach for image sensing with smartphones", IEEE Trans. on Mobile Computing **15**, 5, 1249–1263 (2016).

Xiang, C., P. Yang and S. Xiao, "Counter-strike: Accurate and robust identification of low-level radiation sources with crowd-sensing networks", Personal Ubiquitous Comput. **21**, 1, 75–84 (2017).

APPENDIX A

IOT STANDARD AND PROTOCOLS

As an emerging field, IoT protocols are evolving to better adapt to the needs of the industry. Broadly these protocols are divided into two main categories, Data Protocols, and Network protocols. As the devices and IoT sensors are distributed, communication and data protocols help to establish and maintain the IoT application.

A.0.2 Network Protocols

1. HyperText Transfer Protocol (HTTP): It is the basic protocol that is used for communication for large scale IoT framework. For low constraint devices, with limited battery, data, and resources, this protocol is not preferred.

2. Long Range Wide Area Network (LoRaWAN): It is a long-range wireless connection protocol which is used in either private or global network. It is used on a large scale where millions of devices may be connected and function with low power and memory.

3. 6LoWPAN: This protocol uses IPV6 Low Power Wireless Personal Area Network. Data compression and header information allow IPv6 to be used for communication over a range of networks e.g. Local, Metropolitan and Wide Area Network.

A.0.3 Physical Layer

1. Near Field Communication (NFC): NFC allows transmission of data between nearby devices in range on 4cm. The transmission occurs without contact. Example of NFC is payment systems and keycard.

2. WiFi: This technology connects the devices over wireless connections using standard IEEE 802.11 standards.

3. RFID: This technology makes use of radiofrequency to communicate. RFID chips store and maintain inventory.

4. LPWAN: Used for data transmission over long distances with low power.

5. ZigBee: Allows communication between smaller devices with low power and low data rates within a small distance. Example of ZigBee is smart homes or electric meter monitoring.

6. Bluetooth: It allows communication with nearby devices in a range of 100cm. A low energy feature allows us to save battery energy and is useful for wearable devices.

A.0.4 Data/Application Protocols

1. Message Queuing Telemetry Transport (MQTT) protocol: It works on a publish-subscribe model to offer low power consumption and minimized packet. It is particularly used in the industrial domain earlier known as SCADA protocol.

2. Advanced Message Queuing Protocol (AMQP): This protocol allows message queuing and routing in a secure and reliable way. Three stages of AMQP are Message, Queue Exchange and Binding. E.g. in the banking app.

3. Constrained Application Protocol (CoAP): It is specially used for low power and low memory devices where communication times and resources on the devices can be optimized. The communication is based on UDP protocol sending binary data in Efficient XML Interchanges (EXL).

APPENDIX B

ENERGY AWARE MAPPING ON MANY-CORE DEVICES

B.1 Approach for Application Colocation Problem

Many-core devices can support applications that require a large number of computations that may not run on a mobile device. To utilize the many-core platforms for maximizing the throughput of the applications, the code in the application needs to execute in parallel. While throughput is a critical requirement for real-time processing, running these applications in data centers incur energy consumption. Mapping the application on the specific many-core platform can reduce the operational cost of a data center by energy savings. Given constraints of throughput of certain applications, this work focuses on improving the performance and energy consumption by a.) Reducing the interference between colocated applications. b.) Mapping the parallel applications on architecture.

The approach is to exploit the parallelism of an application at thread-level and data level and spatial and temporal locality as well as overlapping the computation by communication time of data while processing on many-core devices. By matching the architecture and application parallelism this work proposes to extract maximum throughput from the many-core systems. This study involves the colocation of high throughput medical applications in a hospital scenario. Two types of model predictive controllers are considered 1. Pharmacokinetic model controller 2. Spatiotemporal model controller. The resource requirements of these applications e.g. computational, memory requirements are different and they form the design parameters for evaluation of the multi-core platform.

B.2 Exploiting the Architecture for High Throughput Applications

Table B.1: Multiprocessor Platforms for Fast Processing of Medical Applications.

Core Architecture Model No	i7 ivybridge 3770K	GPU GTX 680	XEON Phi 3120P
Core Frequency	3.5GHz	1.006GHz	1.1 GHz
Num Cores	4	1536	57
HW-thread/Core	2	512 (max)	4
Total Last Level Cache	8.192 MB	512KB	28.5MB
Main Memory	16.3815GB	2.048 GB	6GB
Total Power (TDP)	77 W	195W	300 W

This table describes the details of the architectural features of multi-core platforms that are exploited for the fast processing of different types of parallel applications. Design space is based on the resource constraints of each platform. Table B.1 gives a summary of the architectural details of the many-core platforms used in this study.

Both the MPC applications are executed in parallel by exploiting the architecture for thread and data-level parallelism. The simulation of the model generates an output which is used to change the level of drug-infused. These MPC applications are run on the different architectures and evaluated for Flops/Joule. This Flops/joule metric is further used to obtain the most energy-efficient scheduling.

Pharmacokinetic Model Application takes input as a number of patients that are monitored. The number of threads is generated based on the number of patients. For each patient, the application predicts the value of the drug level for the next half hour. Once the predicted value is obtained, the controller compares this value to the reference drug level and adjusts the signal to the actuator of the infusion pump.

Spatio-temporal Diffusion Model Application input consists of a number of patients and the size of the tissue (grid size). The model is computed to predict the output of drug content in the tissue. Based on this predicted value, the controller compares it to the reference drug level and outputs the signal to the actuator of the infusion pump.

B.3 Performance of Energy-Aware Application Mapping Scheme

The spatiotemporal or the Pharmacokinetic application has a different performance and energy consumption when running on different multicore platforms. The results indicate that Pharmacokinetic application will run more efficiently and fast on i7 while the Spatiotemporal application performs better on the MIC. In the hospital scenario, different patients may be monitored with a different application based on the criticality and need of the patient. In the case of different applications that are monitoring multiple patients at a given time, can we utilize the existing heterogeneity of platform e.g. i7 with MIC card to gain more energy savings and better performance?

Consider a scenario, where we have a sample set of 3 patients that are monitored using the Pharmacokinetic model application and 16 patients are monitored using the Spatiotemporal diffusion model application. This sample set of applications is running simultaneously on different platforms. Figure B.1 shows a plot of total execution time and energy consumption of our sample application set when running on the platform. Note that the energy consumption shown is inclusive of the idle energy. As Pharmacokinetic application performs well on i7 and the Spatiotemporal application performs well on the MIC we use this combination. The last value in Figure B.1, indicates that MIC-i7 gives better performance and energy savings.

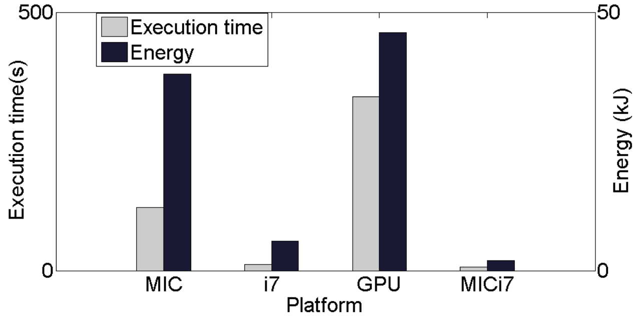


Figure B.1: Sample Run Showing Energy and Performance of 3 Patients Monitored by Pharmacokinetic Model and 16 Patients Monitored by Spatiotemporal Model Application.

APPENDIX C

ELASTIC VM MANAGEMENT SCHEME

The dynamic VM management system is an extension to OpenStack, an open-source cloud platform. In its current offering, OpenStack trades the energy for swiftness through static resource assignment to VMs. In the scenario of Intel Base Station applications that service LTE workloads, a pattern of VM usage is observed. These VMs are hosted on Cloud Radio Access Network (C-RAN), which provides virtualization of resources and Network Function Virtualization to support the flexibility and scalability in MEC hosted applications in 5G.

C.1 MEC Infrastructure

MEC architecture is widely being used especially with 5G networks. MEC enables to deploy apps on the edge of the mobile network. It supports processing edge data with low latency, real-time processing with privacy: Figure C.1 shows the components of the server. It includes a virtualization component that supports the virtualization of the network, resources for MEC apps. The network virtualization provides isolation to MEC apps using Software-defined networking, (SDN) and Network Function Virtualization (NFV). High-speed networks in 5G and a large number of connected devices, as well as data obtained, can be supported by a Cloud radio access network or C-RAN.

Given the time-varying nature of the cloud applications' workload in Base Station Applications, elasticity can be infused into the cloud platform to scale the active resources according to the VMs' input workload and save energy by removing unnecessary idle power. Proactive VM management solution dynamically decides on the VM's resource assignment and consolidates the VMs into a fewer number of hosts considering the VMs' future resource requirements and VMs' migration overhead. Dynamic VM management is proposed as an optimization problem and a heuristic solution is proposed to solve it.

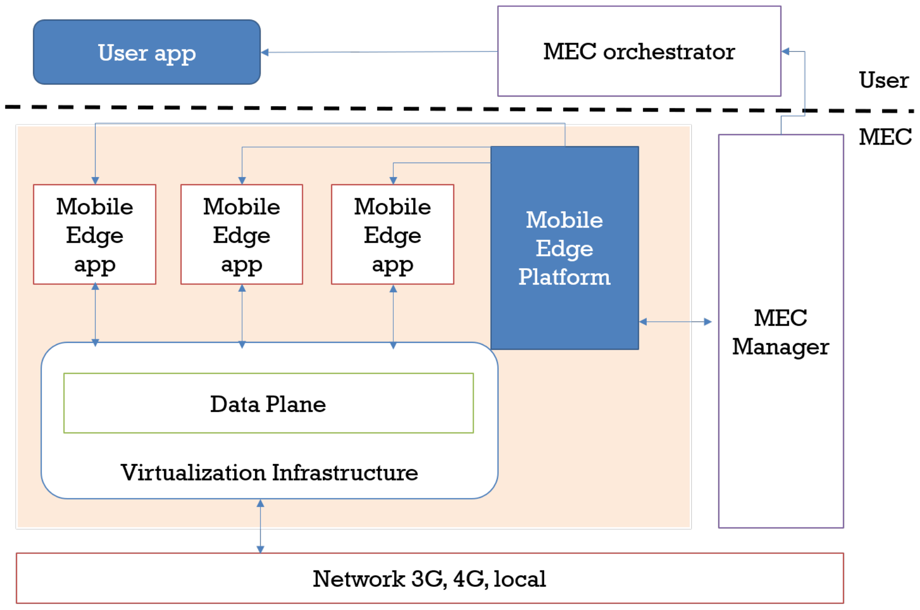


Figure C.1: Components of MEC Server.

The proactive VM management scheme is designed to consolidate VMs by addressing the overload and underload conditions on servers based on the predicted workload. This scheme is evaluated on OpenStack Cluster on real-world Internet traces and Intel travel-box clusters. The experimental results show that the VMs' future resource requirements can be predicted with reasonable accuracy (15% prediction error) and that they are successfully utilized to detect underloaded and overloaded hosts.

The challenges in dynamic VM management are as follows:

1. Maintaining the quality of service of VMs
2. Optimal scheduling needs VM prediction with future values known over a long prediction window.
3. Fine variations in workload can cause performance degradation
4. Consolidation of VMs using the migration of VMs retaining the current state of VMs i.e. the migration overhead costs.

C.2 Approach

Consider a discrete-time model by dividing the time into equal intervals, called slots, where the dynamic VM management policies can be updated. The slot duration has to be short enough to capture the workload variabilities, yet long enough to predict the workload with reasonable accuracy and to avoid VM migration overhead.

At the beginning of each slot, the dynamic VM management solution determines VM migration policies by reconciling a number of competing objectives, e.g., increasing the VM consolidation rate and consequently removing unnecessary idle power, maintaining the quality of service requirements of the VMs' applications, and reducing number of VM migrations to avoid VM migration overhead. We consider that the OpenStack controller node determines the dynamic VM management policies i.e., VM migration and that the cloud platform (OpenStack) is capable of dynamic VM migration.

Given a long period consisting of T slots, a cloud with N nodes each associated with a limited capacity of R_i and the energy consumption $p_{idle,i} + p_i$, time-varying number of VMs, $M(t)$, each associated with time-varying resource requirements of $r_j(t)$, and VM migration overhead of $q_{i,j}(t)$ and $q'_{i,j}(t)$, find VM placement policies over time (which VM to migrate, when to migrate and were to migrate) so that to minimize power consumption, while maintaining the VMs' resource requirements. The assumption is that each VM resource r, as well as VM migration overhead q and q', is modeled as a vector of CPU, RAM, Memory and Network resources.

C.2.1 Overview of the Heuristic Solution

The complexity of solving the aforementioned optimization problem is two folds: (i) characterizing and predicting the parameters over time T (i.e., predicting resource $r_i(t)$, $q_{i,j}(t)$, $q'_{i,j,t}$, and p_i over T) , and (i) finding a computation-efficient solution to solve the problem over T with near-optimal performance. In particular, developing a computation efficient solution of the above dynamic VM management problem require resource usage profiling and modeling of Internet applications and devise a heuristic solution to dynamically characterize the parameters and find a dynamic VM placement policy to reduce power consumption. The heuristic solution instead of forecasting the exact resource requirements of VMs estimates resource requirement boundaries to ensure their quality of service. The following describes the basics of the heuristic solution:

- Given daily basis variation of Internet application, witnessed by several researchers, T is the number of slots for a day.
- To characterize resource requirements of VMs we forecast their average resource utilization (i.e., CPU utilization, the average number of send/receive packets) and make use of a performance model that maps the average resource utilization to their resource requirements such that their quality of service is maintained.
- To estimate the available free capacity of compute nodes, we use performance models that detect overloading and under-loading conditions of compute nodes. The overloading hosts need to reduce their number of VMS and the VMs of the under-loaded host can be consolidated.
- At this stage of the project, we do not exactly characterize VM migration overhead instead the heuristic solution avoids frequent VM migration and performs migration when the source and the destination have sufficient free resources (i.e., they are not in the overloading conditions). In this way, there are enough resources to address migration overhead without affecting the performance of the hosted VMs.
- Instead of brute-force search to find the optimal VM assignment policy, we linearly search through computing nodes over T, detect overloading and underloading host and determine the VM migration polices.

C.3 Elastic VM Management Scheme

We implement a software application which can be considered as an extension to the existing OpenStack framework. As shown in Figure C.2, the application is compatible with the client-server architecture of OpenStack and consists of several components that either reside in the controller node or the compute nodes as follows.

The system application is split into two main parts, resource monitoring that collects the resource parameters such as CPU utilization of the host VMs and processes it in the controller, and dynamic VM management algorithm which decides on the VM placement policies in the controller. Once the VM is chosen for migration, the decision is integrated by changing the input of destination hosts in the nova-scheduler. A detailed explanation of the above components is given in the following subsections.

C.3.1 Resource Monitoring

The resource monitoring tool collects the resource usage over very short intervals (e.g., seconds) which are used to calculate the statistical information of the VMs' resource usage over slots. The tool has two components. One component resides in

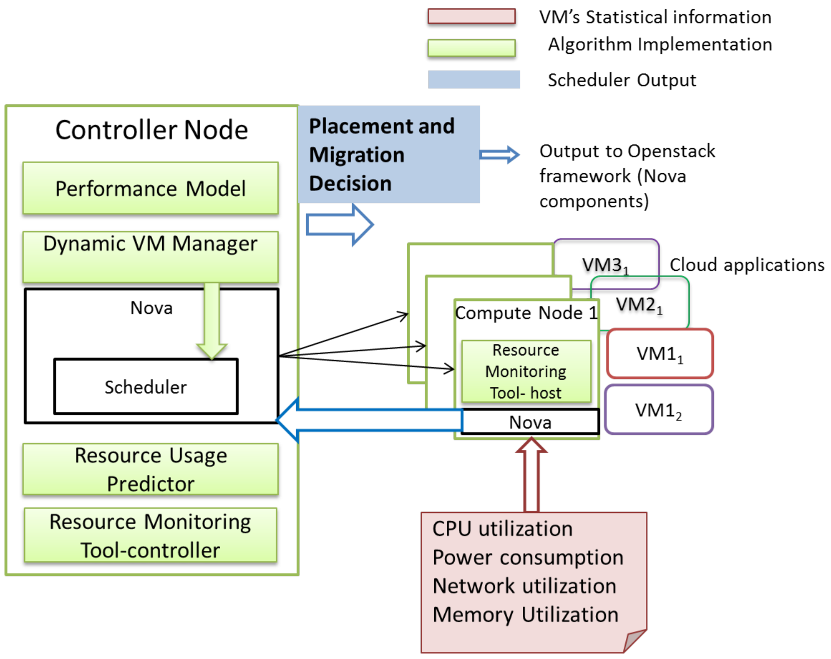


Figure C.2: Architecture for Dynamic VM Management Scheme.

each active compute node that collects the resource usage parameters of each VM including CPU utilization (i.e., number of vCPU, vCPU time), network utilization (i.e., packed send/receive and bytes sent/received), memory (i.e., memory size and memory usage) over seconds using libvirt. This information is stored in a database in the controller. The other component resides in the controller and computes the statistical parameters of VMs' resource usage over slots. Such statistical information which includes mean, standard deviation, minimum and maximum of all resource parameters are then stored in the database and are used to train and update the resource prediction model.

C.3.2 Resource Usage Predictor

The Predictor tool is designed by integrating the R statistical tool into python known as the "rpy" library. Based on prediction results in § 5.2, SARIMA is used to predict the resource usage of VMs that varies according to the workload. The R statistical library allows automation in building a model based on sufficient training data. The prediction model uses training data of each VM available in the database and predicts the resource usage of each VM over a prediction window (e.g., 24 hours). We use the forecast library from R to build a SARIMA prediction model. The prediction model is built separately for each resource usage parameters (e.g., CPU utilization and network utilization). Once the model is built, it can use the trace of resource utilization up to current time to predict the resources over the prediction window of 24 hours. Further, models are updated periodically (with configurable period length) to capture the resource usage dynamics.

The performance model describes the conditions where the resource assignments of the VMs are sufficient to maintain their quality of service and are used to evaluate overloading and underloading conditions of a compute node for deciding on the VM consolidation/dissemination.

VM Application Performance Model

We assume that the quality of service requirement of the VM applications is tied to the resource utilization of the VMs. In particular, we consider that by imposing a cap on the CPU utilization level of the VM application,the quality of service degradation of the application is avoided. This is witnessed by the related work and by our previous research results.

Compute Node Performance Model

When VMs are running simultaneously, the resource utilization of underneath host is the cumulative sum of resource utilization incurred by individual VMs (refer § 6.1) + interference. VM management decisions are based on the following thresholds.
Upper Threshold, U_{th}: This value is the maximum value of total resource utilization incurred by VMs such that the VM applications' quality of service is maintained. If in any slot, the total CPU utilization with a standard deviation of the VMs exceeds this U_{th}, the VMs do not have sufficient resources, the host is detected to be overloaded. Figure C.3 shows that the cumulative CPU utilization exceeds the U_{th} in point X.
Lower Threshold, L_{th}: This is the maximum value of cumulative CPU utilization such that if the total resource utilization incurred by VMs with their standard deviation is below this value, the host is detected to be underloaded at that time. L_{th} is based on the consolidation rate and migration overhead. Lower threshold results in more consolidation and more migrations. Figure C.4 shows the cumulative CPU utilization is maintained below L_{th}, hence the host is detected to be underloaded.

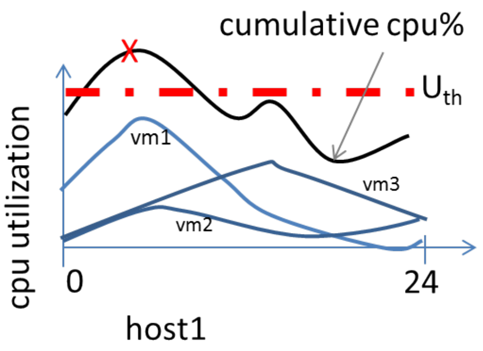


Figure C.3: Overload Host

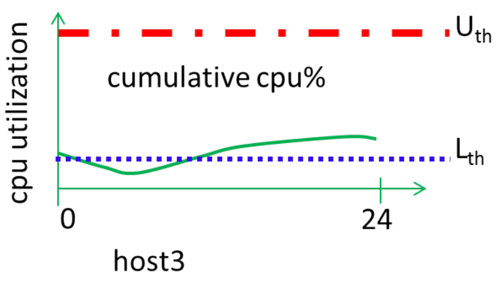


Figure C.4: Underload Host

C.3.4 Dynamic VM management

The heuristic VM management algorithm that adjusts the active servers according to the input workload of VMs. Figure. C.5, describes the steps of our algorithm.

In every slot (hourly), the resource usage parameter for each VM is predicted over T (i.e. next 24 hours for $T = 24$). The dynamic VM manager uses the predicted resource usage and the performance model to detect the overload and underload conditions of each compute node. In case, overload is detected for any node, the most responsible VM for the overload is chosen to migrate. In case none of the hosts are overloaded, the underload condition for the host is checked. If a host is underloaded, all the VMs from that host are migrated to other hosts by choosing a new destination host. When a VM is chosen for migration, the new host is selected from available hosts such that considering the new VM, its migration overhead and the existing VMs, the host does not get overloaded.

The details of this algorithm are discussed below.

Detecting Overload Condition

The overload condition of VM occurs when there are not enough resources on the host to meet the quality of service requirements of the workload. With the knowledge of predicted CPU utilization for the next 24 hrs, for each VMs on the host, if the performance model (refer §C.3.3) detects overload in any of the 24 hours in the future, the host is detected to be overloaded. We note the time at which maximum overload occurs. As shown in Figure.C.3, maximum overload is seen to occur in the X slot.

Choosing which VM to Migrate

The slot in which overload is estimated to occur i.e. X slot is is considered while choosing a VM to migrate. With several VMs running on the host, the VM that is most likely contribute to overload condition is detected by comparing the CPU utilization contributed by individual VMs at the X slot in the future. The VM that has a maximum CPU utilization at the X slot is chosen as a candidate for migration. As shown in Figure.C.3, vm_1 is chosen for migration.

Finding a New Host for the VM

A host is chosen as a candidate for a new VM if, by the addition of a new VM, all the VMs have enough resources to meet the quality of service of VMs' application. Using the predicted data for the new VM and the predicted resource usage of the VMs already residing on the host, a new host is found such that the performance model (refer §C.3.3) detects no overload in T slots in the future, see Figure C.4. From the available hosts, the first host that can accept the new VM is chosen as the destination host for the selected VM.

When to Migrate the VM

We choose to migrate a VM when predicted VM utilization and the VM migration overhead does not cause overhead while migrating from the source host to the destination host. The VM migration overhead is not exactly characterized but it is ensured that the source and the destination have sufficient free resources (i.e., they are not in the overloading conditions) by overestimating the VM migration overhead when migrating a VM. In this way, there are enough resources to address migration overhead without affecting the performance of the hosted VMs. Starting from the slot, when the overload occurs (i.e. X slot) towards the current slot, migration slot is chosen such that the VMs' predicted utilization and VM migration overhead does not cause overload on the source and destination hosts.

Detecting the Underload Condition

The underloading condition exists when existing VMs are predicted to have abundantly extra resources than required to meet the quality of service of VMs' application. In such a case, it is more efficient to move the VMs to other active hosts and power of the underloaded host. We estimate the underloading to occur based on VM's CPU utilization as an indication of host utilization. With the knowledge of predicted CPU utilization for next 24 hrs, for each VMs on the host, if the performance model (refer §C.3.3) detects the host to be underloaded at all the times in the future, all the VMs are migrated to other hosts by finding a suitable host.

To prevent frequent migrations and reduce the migration overhead, the condition of overload is evaluated and then checked if underload exists. In some cases, neither overload or underload may exist as seen in Figure C.4.

C.4 Performance of VM Management Algorithm

The proactive VM management solution is evaluated using realistic traces and systems. Adapting SARIMA to the online prediction of resource usage gives a prediction accuracy of 15% for the next hour and increases to 32% over the prediction window of 24 hours. The prediction results are used by our performance models to detect overload and underload of hosts. The host CPU utilization is used as an indicator of the VM application's performance. Considering the internet workload, the cumulative CPU utilization incurred by individual VMs is related to host CPU utilization. This can be used to detect the overload and underload condition of the hosts. The VM migration overhead is also considered as a job to ensure that VM application performance is maintained during the VM's migration. The feasibility of our proactive VM management scheme is based on predicted VM resource usage, VM migration overhead, and threshold-based VM management algorithm to improve the consolidation rate of the VMs by extending the existing OpenStack. The power savings in the cluster were observed to be 23% compared to the original scheduling algorithm in OpenStack.

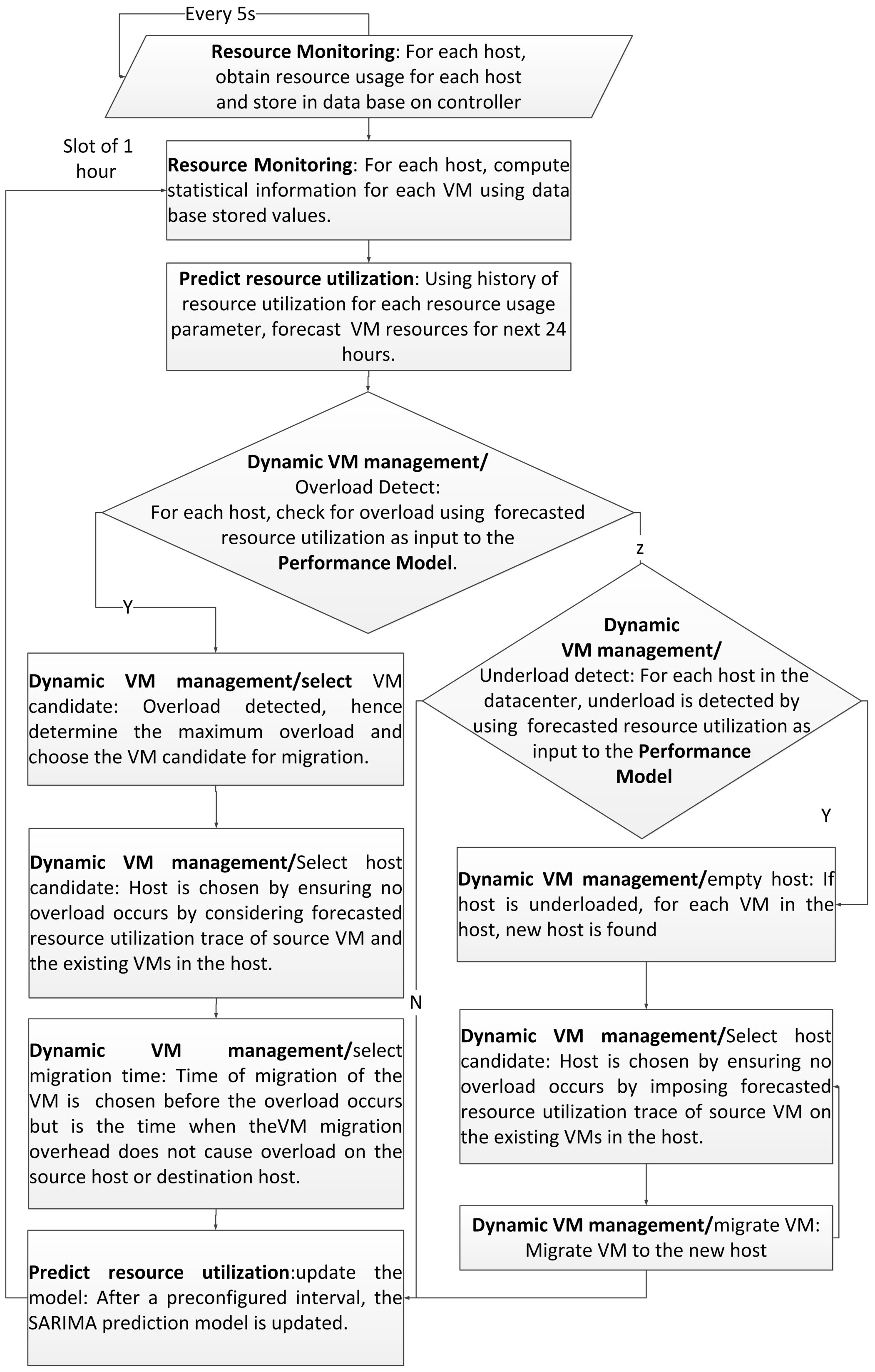

Every 5s
Resource Monitoring: For each host, obtain resource usage for each host and store in data base on controller
Slot of 1 hour
Resource Monitoring: For each host, compute statistical information for each VM using data base stored values.
Predict resource utilization: Using history of resource utilization for each resource usage parameter, forecast VM resources for next 24 hours.
Dynamic VM management/ Overload Detect: For each host, check for overload using forecasted resource utilization as input to the **Performance Model**.
Y
z
Dynamic VM management/select VM candidate: Overload detected, hence determine the maximum overload and choose the VM candidate for migration.
Dynamic VM management/ Underload detect: For each host in the datacenter, underload is detected by using forecasted resource utilization as input to the **Performance Model**
Y
Dynamic VM management/Select host candidate: Host is chosen by ensuring no overload occurs by considering forecasted resource utilization trace of source VM and the existing VMs in the host.
Dynamic VM management/empty host: If host is underloaded, for each VM in the host, new host is found
N
Dynamic VM management/select migration time: Time of migration of the VM is chosen before the overload occurs but is the time when theVM migration overhead does not cause overload on the source host or destination host.
Dynamic VM management/Select host candidate: Host is chosen by ensuring no overload occurs by imposing forecasted resource utilization trace of source VM on the existing VMs in the host.
Dynamic VM management/migrate VM: Migrate VM to the new host
Predict resource utilization:update the model: After a preconfigured interval, the SARIMA prediction model is updated.

Figure C.5: Algorithm for Dynamic VM Management.