Efficient Schemes for Crowd-sensing based Applications

Thesis submitted

by

Arpita Ray

DOCTOR OF PHILOSOPHY (ENGINEERING)

Department of Computer Science and Engineering
Faculty Council of Engineering & Technology
Jadavpur University
Kolkata, India

2024

Efficient Schemes for Crowd-sensing based Applications

Thesis submitted

by

Arpita Ray

DOCTOR OF PHILOSOPHY (Engineering)

under the supervision of

Prof. Dr. Sarbani Roy

Department of Computer Science and Engineering

Jadavpur University

Kolkata, West Bengal-700032, India

Dr. Chandreyee Chowdhury

Department of Computer Science and Engineering

Jadavpur University

Kolkata, West Bengal-700032, India

2024

Index No: 158/19/E

1. **Title of the Thesis**:
   Efficient schemes for crowd-sensing based applications

2. **Name, Designation and Institution of the Supervisor**:

   Dr. Sarbani Roy
   Professor
   Department of Computer Science and Engineering
   Jadavpur University
   Kolkata-700032

   Dr. Chandreyee Chowdhury
   Associate Professor
   Department of Computer Science and Engineering
   Jadavpur University
   Kolkata-700032

1. **List Of Publications**:

   (a) **Journal**:
   Published:

   i. **Ray, A.**, Chowdhury, C., Bhattacharya, S. and Roy, S., 2023, "A survey of mobile crowdsensing and crowdsourcing strategies for smart mobile device users," *CCF Transactions on Pervasive Computing and Interaction*, 5(1), pp.98-123. (2022 SCI Impact Factor: 2.109)

   ii. **Ray, A.**, Chowdhury, C., Mallick, S., Mondal, S., Paul, S. and Roy, S., 2020, "Designing energy efficient strategies using markov decision process for crowd-sensing applications", *Mobile Networks and Applications*, 25, pp.932-942. (2022 SCI Impact Factor: 3.8)

   iii. Chowdhury, C., Roy, S., **Ray, A.** and Deb, S. K., 2020, "A fault-tolerant approach to alleviate failures in offloading systems", *Wireless Personal Communications*, 110(2), pp.1033-1055. (2022 SCI Impact Factor: 2.2)

   (b) **Conference**:

   i. **Ray, A.**, Mallick, S., Mondal, S., Paul, S., Chowdhury, C. and Roy, S., 2018, December, "A framework for mobile crowd sensing and computing based systems", in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)* (pp. 1-6), Indore, India.

   ii. **Ray, A.**, Chowdhury, C. and Roy, S., 2017, December, "Strategic decision for crowd-sensing: An approach based on Markov decision process", in *2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)* (pp. 1-6), Bhubaneswar, India.

   iii. **Ray, A.**, Prasad, R., Chowdhury, C., and Roy, S., 2023, "Energy Efficient Task Execution Through Edge Federation Utilizing Simulated Annealing", in *Proceedings of International Conference on Security, Surveillance and Artificial Intelligence, ICSSAI 2023.* Part 3, Ch 25, pp.(247-257), Kolkata, India.

   (c) **Book Chapter**: None

2. **List of Patents**: None

3. **List of Presentations in International Conference**:

   (a) **Ray, A.**, Mallick, S., Mondal, S., Paul, S., Chowdhury, C. and Roy, S., 2018, December, "A framework for mobile crowd sensing and computing based systems", in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)* (pp. 1-6), Indore, India.

   (b) **Ray, A.**, Chowdhury, C. and Roy, S., 2017, December, "Strategic decision for crowd-sensing: An approach based on Markov decision process", in *2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)* (pp. 1-6), Bhubaneswar, India.

   (c) **Ray, A.**, Prasad, R., Chowdhury, C., and Roy, S., 2023, "Energy Efficient Task Execution Through Edge Federation Utilizing Simulated Annealing", in *Proceedings of International Conference on Security, Surveillance and Artificial Intelligence, ICSSAI 2023.* Part 3, Ch 25, pp.(247-257), Kolkata, India.

Certificate from the Supervisor

This is to certify that the thesis entitled ***"Efficient schemes for crowd-sensing based applications"*** *submitted by* ***Ms.Arpita Ray****, who got her name registered on* **3rd June 2019**, *for the award of Ph.D.(Engineering) degree of Jadavpur University, is absolutely based upon her own work under the supervision of* ***Prof. Dr. Sarbani Roy*** *and* ***Dr. Chandreyee Chowdhury*** *and that neither her thesis nor any part of the thesis has been submitted for any degree/diploma or any other academic award anywhere before.*

Sarbani Roy. 06/05/2024

Dr. Sarbani Roy,
Professor,
Department of Computer
Science and Engineering
Jadavpur University.
(Supervisor)

Professor
Computer Sc. & Engg. Department
Jadavpur University
Kolkata-700032

Chandreyee Chowdhury 06/05/2

Dr. Chandreyee Chowdhury,
Associate Professor,
Department of Computer
Science and Engineering
Jadavpur University
(Supervisor)

ASSOCIATE PROFESSOR
Dept. of Computer Sc. & Engg.
JADAVPUR UNIVERSITY
Kolkata - 700 032

Statement of Originality

I, Ms. **Arpita Ray** registered on **3rd June, 2019** do hereby declare that this thesis entitled **"Efficient schemes for crowd-sensing based applications"** contains literature survey and original research work done by the undersigned candidate as part of Doctoral studies.

All information in this thesis have been obtained and presented in accordance with existing academic rules and ethical conduct. I declare that, as required by these rules and conduct, I have fully cited and referred all materials and results that are not original to this work.

I also declare that I have checked this thesis as per the **"Policy on Anti Plagiarism, Jadavpur University, 2024"**, and the level of similarity as checked by iThenticate software is **5%** .

Signature of Candidate: Arpita Ray
Date : 6/5/2024

Certified by Supervisor(s):

(Signature with date, seal)

1. Sarbani Roy 06/05/2024

Professor
Computer Sc. & Engg. Department
Jadavpur University
Kolkata-700032

(Dr. Sarbani Roy,

Professor,

Department of Computer Science and Engineering

Jadavpur University.)

2. Chandreyee Chowdhury 06/05/24

ASSOCIATE PROFESSOR
Dept. of Computer Sc. & Engg.
JADAVPUR UNIVERSITY
Kolkata - 700 032

(Dr. Chandreyee Chowdhury,

Associate Professor,

Department of Computer Science and Engineering

Jadavpur University.)

Dedicated

To my mother and my daughter

To all the researchers

Contents

List of Figures

List of Tables

Acknowledgment

First and foremost, I would like to convey my heart felt gratitude and thanks to my two thesis supervisors, Prof. Dr. Sarbani Roy, Professor, Jadavpur University and Dr. Chandreyee Chowdhury, Associate Professor, Jadavpur University for their ceaseless support in my PhD study and research, for their never ending enthusiasm, motivation and patience and their immense knowledge which they have imparted on me during my course tenure. Their guidance have assisted me immensely in my research and in writing of my thesis. It is impossible for me to envisage better advisor and mentor combination than them for my thesis work.

Besides my supervisors, I would like to sincerely thank the rest of my thesis committee members for their timely productive and valuable opinions to enhance the standard of my thesis work. My sincere thanks also goes to the Computer Science and Engineering Department, Jadavpur University, to provide me with an excellent work environment.

I would also like to thank my junior Asif who has endlessly assisted me in writing process of my thesis. My special thanks goes to my dear friends Niladri, Ayan, Ruby and Paramita for always propelling and encouraging me to stay focused on my work. I also want to thank my family, Raja, Baba, Bordi and my daughter Juin who have put up with me during my not so good days and have always loved me and supported me unconditionally.

Last but not the least, I am especially grateful to my Ma for whatever little I could achieve. I want to thank my dear Ma, for her love, support, generosity and endless patience she has bestowed upon me. Without her, this thesis could not have seen the light of day. She had been my pillar of support all through my journey from struggle to strength. Since I lost her to Covid, last two years had been extremely difficult. I shall always seek her support in my pursuit of excellence endowed upon me by her divine benevolence.

Abstract

In recent times, smart handheld devices have become an indispensable part of human lives. These devices have inbuilt sophisticated sensors which are sensing the environment and hence a huge amount of data are being accumulated in the devices. These data can be utilised for improving the lives of the people. The various applications of mobile crowd-sensing (MCS) are public safety, traffic and transportation planning, environment monitoring to name a few. Along with these implementations comes a set of challenges. The main challenges in MCS: i) Energy efficiency of the smart devices, ii) Fault tolerance of the system when the data are being offloaded to the backend servers and iii) Computation and load balancing while distributing the offloadable tasks to the backend servers in an efficient manner. These are the main challenges addressed in this thesis. The first challenge is addressed by proposing energy efficient strategies that enable mobile devices to utilise the energy in a well planned fashion so that the limited resources can be managed well without hindering the quality of services. In order to address the second challenge, an efficient framework is proposed which helps in detecting faults, segregating the faults and trying to steer clear of the faults whenever possible. The framework also aids in tolerating faults. The third challenge is addressed by implementing optimisation technique, Simulated Annealing (SA) for allotting the computation intensive and time-sensitive tasks to the backend servers in a balanced manner.

All the experiments in this thesis are simulated and the results of simulation are validated by real-life implementations. In a situation where the smart handheld devices needs to take the decision of whether to participate in crowd-sensing, energy efficient strategies are proposed which outperforms the benchmark strategies in better utilisation of the energy. In the event of finding faults in offloading systems, our proposed framework not only detects faults in the offloading systems, but also classifies the faults. The framework also helps to tolerate one degree fault. It has also been observed in offloading systems, that during dissemination of tasks to the backend servers, our proposed framework, implementing SA outperformed to other techniques with 90% success rate for efficient load balancing and completion of the allotted task. These proposed framework can be utilised in mobile devices for better utilisation of the constricted resources.

Chapter 1

Introduction

1.1 Overview

In the past few years, there has been tremendous growth in the field of wireless and mobile communication. Better connectivity has been provided to the billions of smart handheld devices as the networks have become more efficient and highly capable to handle the high amount of data. With the increase in the number of sensors such as cameras, gyroscopes, accelerometers in most smart handheld devices, there has been a constant effort to fit in more storage capabilities and processing power in these devices. This has led to the emergence of a new paradigm called mobile crowd-sensing (MCS), where the objective is to collect geospatial data locally and share this data with the wider community.

Hence, mobile crowd-sensing is defined in [1] as a category of applications where individuals with sensing and computing devices collectively share data and extract information to measure and map phenomena of common interest. Mobile crowd-sensing can be very useful in sharing the knowledge which is available locally to implement in various day-to-day applications for improving the quality of life. The various benefits of mobile crowd-sensing can thus be extended to public safety, traffic and transportation planning, environment monitoring and urban dynamic sensing, location services, mobile social recommendation and healthcare [2]. The smartphones help in the collection of the useful information and this information is uploaded and kept in the server. Later, this data can be utilized with complex computing in order to extract meaningful information. Hence, the computing is done in the backend servers where the crowd sensed data is being uploaded. Often, for time-sensitive data and few other applications where data needs to be analysed at the user end, crowd plays an important part. The term "crowd computing" can be defined as a technique where the distributed tasks are computed either by utilizing the computational capability of smartphones carried by the crowd or by using the human intelligence of the crowd. Thus, the smartphones play the role of processing tasks often utilizing human intelligence in crowd computing. So integrating the power of sensing of smart mobile devices and the human intelligence, significantly allows the mobile devices to contribute in the process of crowd computation, when the device has gained sufficient knowledge. So, a framework is needed which can apply the expertise of crowd and utilize the data that are sensed by the mobile devices for performing a task without offloading the task to the backend servers, and provide a reliable solution to the user in a very short time frame [3].

Though mobile devices have various sensing capabilities such as accelerometers, global position systems (GPS), cameras, microphones which help them to capture data from the surrounding environment, there are a number of various issues that needs to be taken into account. These are: limited battery life of the devices, availability of the devices for sensing with respect to its current load. Besides, the incentive to be given for participating in sensing and sharing of the sensed data influences the factor of device availability. This in turn affects the reliability of crowd-sensing as well. Among all these factors, utilization of energy in an efficient manner plays a key role as these devices come with limited battery power (in order to make the device portable). Hence, the device should be made intelligent enough to take the decision of which jobs to execute and when to stop the device from executing any task [4], [5].

Due to the limited resources of the smart handheld devices, it often becomes necessary to offload the computation intensive tasks to backend servers which have better processing capabilities so that the smart handheld devices can remain active for longer duration of time. Various methods have been proposed to offload the tasks [6], [7], [8] but not many works [9], [10] has been done in scrutinizing the different types of faults which happens during offloading of the tasks and ways to eliminate those faults. So, a framework is necessary to detect and prevent the faults or tolerate the fault during offloading [11].

Tasks which are time-sensitive and computation intensive in nature needs to be offloaded to the backend servers such as, cloud. This significantly reduces the load on the resource constrained smart mobile devices, but there are certain issues while uploading the tasks to the clouds. However, offloading tasks to the cloud results in more network bandwidth and less reliability for time-sensitive and location-aware tasks. So, a new paradigm evolved named edge computing, [12] which is a distributed computational paradigm that has been responsible to broaden the cloud services to the edges where the edges form the logical layer positioned as close to the data sources as possible, and it forms the inter-medial layer joining the edge servers to the cloud servers. Interestingly, if the edges are geographically distributed and are not aware of each others existence then these edges can not be made to function smoothly for achieving maximized performance. Thus, a framework is needed to allocate these tasks in a balanced manner to these backend servers which have more computational capacities in an efficient way. An Edge federation framework takes this responsibility of managing these edges and allocate the tasks to the edge servers such that optimized performance from the edges are achieved. A probabilistic technique, Simulated Annealing (SA) has been considered to study the system performance subject to task offloading to a federation of edge servers.

1.2 Research issues and motivation

There are a number of issues that emerged while the smart handheld devices were lend for crowd-sensing. In the following paragraphs few of them are discussed.

- Multi-criteria decision making, ubiquitous computing has let to generation of a large amount of data by the smart handheld devices. These data may often need local analysis in the device itself without offloading the data to the cloud or the edge servers. An integrated framework is needed for analysing the sensed data with the utilization of smart devices and human intelligence for solving a task in a short span. In case, the smart handheld devices are participating in sensing and analysing the data, then an energy efficient strategy needs to be constructed which will enable the mobile devices to decide when to crowd sense and when not to based on certain parameters of the devices such as remaining power of the devices, the load on the devices at that time and the probability of battery recharging.

- Due to limitations of resources in smart handheld devices, often computation intensive tasks can not be executed locally in the devices. Hence, those tasks needs to be offloaded. While offloading the data to the high-computing, resource rich backend servers, often faults are encountered. Occurrence of various types of faults may hinder the performance advantage of task offloading partially or completely. So, there is a need to find techniques to identify faults, to segregate the various types of faults that may be encountered while offloading tasks, and also to tolerate the faults gracefully.

- Task arbitration during offloading the computation intensive tasks to the clouds is the most preferred option for reducing the load of the smart handheld devices, but often the clouds fail to provide the desired Quality of Service (QoS). The power consumed to send data over long distances to the cloud servers or clouds is enormous, and a large amount of bandwidth is consumed in the process while offloading to cloud which often leads to high latency leading to delay in task completion. Often, time-sensitive data that needs to be executed in less time misses deadline due to the high latency. Hence, techniques need to be found which would enable to solve this problem of offloading the tasks as well as distributing the tasks in balanced manner to the backend servers.

All the above issues have been the motivation to construct novel techniques which are summarized in the following section.

1.3 Contributions

The contributions of this thesis comprise of novel approaches to deal with the challenges that are referred in the above section.

- The first contribution proposes a framework which combines the two processes of mobile crowd-sensing and crowd computing and functioning as a unit to process the data locally without the aid of backend cloud servers and providing a reliable solution to the mobile user within a short span of time when a certain amount of incentives are given to the service providers. The results that are computed can be collected both by online procedure when there is internet connectivity or by offline procedure, i.e., with the aid of human intelligence.

- To keep the smart handheld devices active for longer duration of time, an effective strategy is proposed to crowd-sense using Markov Decision Process (MDP), considering possible device conditions prior to deployment. Then, the most suitable strategy, as obtained by solving the MDP formulation, are implemented to work in real-time and the performance of MDP based strategic sensing is compared with other state-of-the-art strategies such as, random or continuous sensing.

- To handle faults during offloading, the faults are identified that may be encountered while offloading the tasks, then the faults are categorised in an offloading systems. An efficient fault tolerant framework is also proposed which is able to tolerate one degree failure.
- As there are a number of issues in offloading the tasks to the clouds, an innovative Edge federation framework is proposed that provides a transparent and seamless services to the end user, by diminishing the latency and optimizing the energy utilization at the same time distributing the tasks to the backend edge servers in a balanced manner. This is done by allocating the task to the fittest edge servers, based on the principle of SA, in case the nearest edge server fails to provide service to the end user.

So, this thesis works towards developing various efficient approaches which aids the smart handheld devices to reduce the energy consumption and utilizes the constrained resources in an optimized manner. The work also proposes framework to detect faults while offloading tasks and ways to balance the tasks while allocating the tasks to the backend servers. The work of this thesis are summarized as follows:

1. An efficient framework is proposed which integrates the two paradigms, i.e., mobile crowd-sensing and crowd computing, to solve problems without involving cloud servers and also proposes an energy efficient strategy applying Markov Decision Process (MDP) to help the smart handheld devices to remain active for longer duration of time.
2. Secondly, an efficient framework is proposed to detect faults, categorize the various types of faults that may be encountered while offloading data to the backend servers and also proposes a fault-tolerant approach to handle the faults during offloading the data.

3. Thirdly, efficient technique is proposed to allocate tasks to the fittest edge servers such that the load is balanced to the resource rich backend edge servers by implementing optimization algorithm such as SA.

1.4 Organisation of thesis

In this thesis, efficient approaches have been formulated and constructed that could assist the smart handheld devices to optimize the utilization of various resources of the devices, that in turn has resulted in helping the handheld devices to operate for a longer duration of time. The organization of the thesis is as follows:

Chapter 2: Literature survey. In this chapter, the background of the various methodologies and the evolution of the various proposed approaches used in the thesis work are discussed.

Chapter 3: A framework for mobile crowd-sensing and computing based systems. This chapter proposes a novel approach integrating the two paradigms, i.e., mobile crowd-sensing and crowd computing, in a framework that addresses both the issues of mobile crowd-sensing and crowd computing at the same time and utilizes the ability of the crowd to solve problems without involving cloud servers in the backend.

Chapter 4: Designing energy efficient strategies using MDP crowd-sensing applications. This chapter proposes an energy efficient strategy applying the MDP by which a smart handheld would crowd-sense while keeping the device active for a longer period of time by aiding the mobile device in taking the decision of when to crowd-sense and when not to considering the remaining energy of the device, it's recharging probability, current computational load, and the incentive it receives.

Chapter 5: A fault-tolerant approach to alleviate failures in offloading systems. This chapter detects and classifies the different types of failures that are encountered while offloading tasks to the backend servers. It also proposes a fault tolerant approach to handling one degree failure. The crash, omission, or transient failures are handled well by this approach. Using historical data to select reliable surrogates from the list of neighbourhood surrogates would enable fault prevention as well.

Chapter 6: Computation and load balancing to allocate tasks in edge computing. This chapter proposes reasonable task offloading strategies to allocate the tasks to the resources in a balanced fashion. Here, the optimization technique, namely SA, has been utilized for implementing task allocation strategies for the edge servers.

Chapter 7: Conclusion and future works. This chapter lays down, in a nutshell the contributions of this thesis, and put forwards the conclusion of the work. It also recognises the potential future research directions.

Chapter 2

Literature Survey

This chapter outlines the essential background literature that is necessary to develop ideas for the comprehensive approaches that have been used in this thesis. The first two Sections 2.1, 2.2 discuss the differences between mobile crowd-sensing and crowd sourcing. The next Section 2.3 discusses the various data sensing and collection frameworks used by crowd-sensing applications minimizing the energy utilization. Offloading systems are discussed in Section 2.4, where the various types of faults in offloading systems are talked about in Section 2.5. It also discusses the various fault-tolerant approaches. The last Section 2.6 of this chapter discusses the various research studies done on offloading tasks to edge servers.

2.1 Mobile crowd-sensing

Mobile crowd-sensing (MCS) [13] refers to the wide variety of sensing models by which individuals collectively share data and extract information to measure and map phenomena of common interest. These systems heavily depend on the rich set of sensors and communication modules that are firmly affixed to the smart devices like smart phones and wearable gadgets [1]. These sensors are used to sense the surroundings, collect the data that is being sensed, and then send the data to the server, which may be present in the cloud or to some other smart devices for further processing of the data. For large-scale sensing and environmental monitoring, a huge number of participants are required to sense the raw data from the surroundings, with the help of sensors that are embedded in the smart devices of the crowd. Based on the type of phenomenon being measured or mapped, MCS applications can be divided into three categories: (a) environmental applications, (b) infrastructure applications, and (c) social applications [1]. The three essential steps for mobile crowd sensing includes data collection, data storage, and data upload. The very first step in MCS is data collection [14], which is broadly divided into the following categories, as shown in Fig. 2.1

- In the first category, it is done through the sensors of the mobile device. Here, the individual user's mobile device may sense data with the user's permission and also sensing data through periodic sampling, or the user is solely responsible for the collection of the sensed data by his mobile device. Also, the collection of context-aware data can be activated by certain contexts that are predefined (e.g., a certain location or a time slot). The user is not responsible for continuously keeping her or his attention on MCS jobs.

- In the second category [15], a large amount of data is collected through mobile social networks.

Context-aware sensing [16] can be achieved in two ways: push and pull. In the case of push-based data collection, the sensing device periodically or instantly sends data to the software module which is responsible for obtaining the sensed data. It enables to issue or endorse a model periodically. On the other hand, in the case of pull-based data collection, the software module, is responsible for obtaining the data that is being sensed, either periodically or instantly, by making a request to the sensors of the smart devices.

There is often redundancy or duplication of the data that is being sensed. As redundancy leads to wastage of the limited resources of smart devices, it hinders the MCS implementation. Data deduplication is a procedure that aids in reducing resource consumption while at the same time improving the QoS (quality of service) of the data that is being sent over the network. In this procedure, the raw data that is being sensed goes through a process of filtering and compression while at the same time maintaining the quality of the data [13], [17]. In this way, the usage of bandwidth is reduced while transferring the data, and less space is used to store the data. In this method, the data obtained after sensing is first divided into blocks, and only the unique blocks are stored. All other incoming blocks are compared with the unique block; if superfluous blocks are encountered, then they are replaced by a reference [18], [19]. Only the unique block and the references are transferred over the network. This way, it helps to reduce the size of the data that is transmitted and bandwidth consumption.

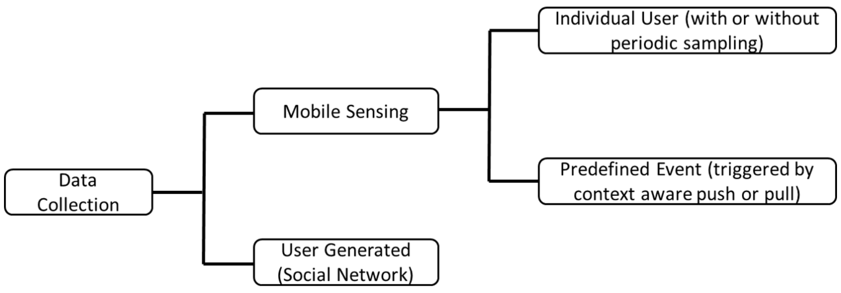

Individual User (with or without periodic sampling)
Mobile Sensing
Predefined Event (triggered by context aware push or pull)
Data Collection
User Generated (Social Network)

Figure 2.1: Different modes of data collection mechanisms for mobile crowd-sensing applications

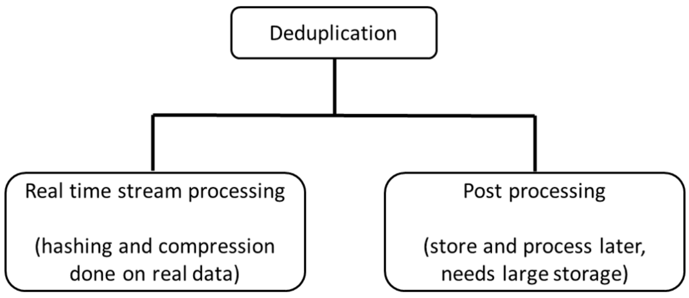

Deduplication
Real time stream processing (hashing and compression done on real data)
Post processing (store and process later, needs large storage)

Figure 2.2: Types of data deduplication based on phase of occurrence

Based on the phase of occurrence of deduplication, it can be classified into two types, as can be seen from Fig. 2.2, real time deduplication [20], [21], [22], and post-process deduplication [23], [24], [25].

In the case of real-time deduplication, at the time of acquiring the data, hashing and compression are done on the raw data. When a fresh set of data is obtained, it is compared with the stored data in the smart devices. If the data is found to be redundant, then it is discarded. Though the storage space gets reduced, the computational load moves from the crowded data processing platform to the terminal, i.e., the mobile device. Hence, it would be a practical problem in situations where the devices have very limited computational capability. In the case of post-process deduplication, the data after sensing is stored in the local mobile device, and then further processing takes place on the sensed data to reduce real-time computational issues. In this method, the device needs to have a larger storage space for storing the huge set of raw data for processing; otherwise, there is a risk of overwriting the stored data when the storage space is smaller.

2.2 Mobile crowd sourcing

Mobile crowd sourcing (MCoS) enables the dispensing of a complex workload to a suitable group of Internet users who are interested in participating in solving a task [26], [27], [28]. This can be achieved by utilising mobile distributed computing and the use of human logical mind when computers fail to decipher a problem. Different users may offer different solutions to solve a particular problem. Fig.2.3 explains how a task is assigned to the crowd worker by the client, how the task is being solved, and how the workers are paid incentives for completion of the task, summarizing the works in [29]. Sometimes a task is divided into sub-tasks and those sub-tasks, are given to the crowd to solve individually. These subtasks are then sent by the crowd to the crowdsourcing platform, where the subtasks are merged to form a feasible solution. This solution is then forwarded to the end user.

Crowdsourcing [30] helps in performing a task faster than an individual, helps in better quality of results, and the results thus obtained are more acceptable as it is supported by a large population of people participating in solving the tasks. For e.g., this website[1] can be used to provide solutions for retail auditing using MCoS. The pitfalls of an in-store can be very well explained if it is performed based on customer feedback about the store.

MCoS helps in getting information regarding customer experience, stock levels of the store, and display compliance all in real time. This crowd sourced data will help in making important decisions for the retail store. Due to the availability of a large number of real time smartphone-enabled shoppers nationwide, it is very easy to obtain data such as photos of the product, get replies to queries based on the sentiments of the shoppers, and so on.

Near real-time data is another factor that helps in predicting the ever changing nature of retail. To be on top and in trend, up-to-date data is most necessary. To make an important business decision, a set of high-quality data is required. The MCoS platform ensures that there is a certain level of standard for the data that is being provided by the crowd. Hence, the MCoS works well for this case as a large variety of replies can be obtained from the customer in real time, aiding in decision-making process with the implementation of the up-to-date changes necessary.

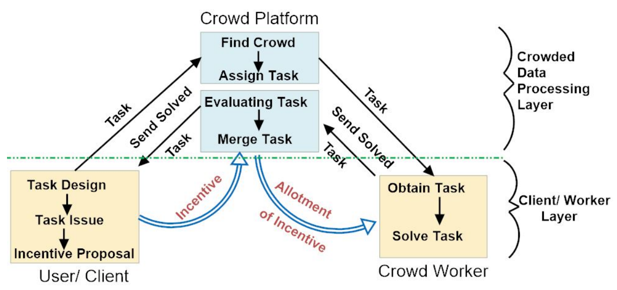


Figure 2.3: Illustration of working of MCoS

Following are a few such benefits of utilizing crowdsourcing for accomplishing a job.

[1]https://wiser.nlm.nih.gov/

- Reduces cost, as instead of paying the professional testers on an hourly basis, the company reimburses only for the bugs that are found. Here, a huge number of participants result in finding more reproducible bugs than a few testers.

- Crowd participation results in diverse, high-quality data that is continually and actively upgraded.

- Due to the availability of different sorts of parameters, crowdsourcing helps in with more thorough and extensive testing.

- Time efficiency due to the multiplicative effort of the crowd.

The outcome of the crowdsourced data may be more acceptable due to the representative involvement of the crowd. The application of MCoS has reached far greater areas than it was thought of while conceptualizing it. Powered by GPS, the smart devices are capable of collecting real time and location-dependent data, which can be implemented in various applications.

For e.g., helping disaster victims by synchronizing the relief program and recording the damages done all in real-time in business with its far reach and flexibility for various applicable proposals. The various application areas of MCoS include social networking and issues [31], [32], environmental monitoring and utilities [33], [34], [35], [36], traffic, navigation, transportation, and urban sensing [37], [38], [39], [40], disaster report [41], [32], [42], translation [43], health monitoring [44], [45] and public safety [46], [47].

The various types of crowdsourcing are as follows from Fig. 2.4

- Collective intelligence: it refers to contribution of an individual's understanding of the crowd.

- Crowd voting: it depends on the judgment made by the crowd. The crowd votes in favor of the best according to the crowd's intelligence. For instance,

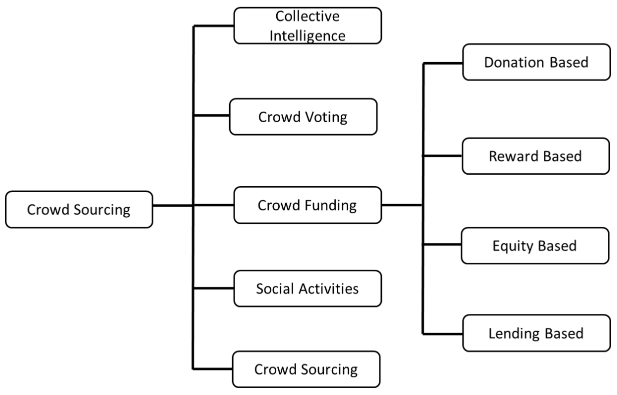


Figure 2.4: Types of crowd sourcing

Yahoo's search produces results performed by crowd voting, as the result is the outcome of the sites that are more popular among the users of the Internet.

- Crowd creation: it refers to the combining endeavors of the crowd to construct a product or service.
- Crowd funding: the main idea of crowd funding is individuals asking for help for a sum of money for a particular endeavor. For example, providing lunches for children in poor countries or providing funding for start-ups.
- Social activities: it refers to human-inspired activities where social interaction by the crowd helps in the collection of information with the help of technology.

A brief summary of the similarity and differences of MCS and MCoS has been put forward and explained in Table 2.1.

2.3 Different data sensing and collection frameworks to reduce energy usage

Various frameworks for mobile crowd-sensing and crowd sourcing have been discussed in this sub-section. Large-scale distributed computation can be done with the help of mobile crowd computing. [48] uses the static task farming method in

Table 2.1: Summary of similarities and differences of mobile crowd-sensing & mobile crowd sourcing

Area	**Mobile crowd-sensing Vs mobile crowdsourcing**
1. Data Collection	In both MCS and MCoS, data collection is through sensors in the device as well as social networks.
2. Privacy & Security	In both MCS and MCoS, concerning challenges are anonymization, data security and privacy.
3. Task Handling	In MCS, task is restricted to sensing of raw data, preprocessing of data by the mobile device. However, in MCoS, tasks comprise of not only sensing and preprocessing of raw data, but also certain other jobs assigned by the cloud platform.
4. User Participation	Both MCS and MCoS motivate user participation for performing tasks.
5. Incentive	In both MCS and MCoS, various incentive mechanisms are adopted to influence the crowd to perform tasks.
6. Human Intelligence	In MCS, human intelligence is not required to perform the tasks but in MCoS often tasks are solved by combining the computing power of the mobile device and human intelligence.

the opportunistic network for message forwarding. But in [49], the authors used work stealing and showed it to perform better than the static task farming method. Various work has been found to be done on mobile cloud [50] and crowd computing [51], [52], where the mobile cloud work mainly concentrates on how the job is offloaded to the powerful super computing devices in the backend, whereas crowd computing is mainly dealing with the collection of sensed data through the various mobile devices or with the help of human intelligence, decisions are taken based on the sensed data.

Honeybee [49] is a programming architecture where the concept of work stealing is implemented to obtain load balancing among the mobile devices that are crowd sensing without previous knowledge of the participating mobile devices. Their work focuses on keeping the participating devices busy most of the time so as to minimize idle time. The job is divided into small, independent tasks so that they can be executed in parallel and save a lot of execution time. Nodes that have finished their allotted jobs try to steal the job from another overburdened node and help in faster execution.

Based on the requirements of the application and the context of the user, the ParticipAct [53] framework finds suitable mobile crowd-sensing policies. This framework takes total responsibility for not only collecting the sensed data properly but also transferring the data to the backend server efficiently. It also helps in processing the data at the server, as well as data harvesting and mining.

The client-side module of the framework not only receives the sensed data, but it also enquires the user whether it wants to compute the data and then efficiently upload the computed data to the server. The server-side module of the framework not only accumulates the data but also manages it and scrutinizes the sensed data. The networking module accepts the data while providing security; it also helps in acknowledging the data that is received properly in the server, resulting in the deletion of the data from the local database of the client module. The post-processor module prepares the data for long-term storage. The data mining module creates the user profile and helps to build the identity of the successful user. This framework also has an administrative module that helps to administer and manage the user profile.

In this work [54], the authors proposed a framework for mobile users and cloud computing infrastructure by merging the service while at the same time optimizing resource usage for mobile crowd-sensing. This service can be amalgamated with different applications on mobile devices. This system not only controls and monitors the sensing task but also enables the upload of the sensed data for further processing in the cloud. This framework also enables to allocate the tasks to mobile devices based on their capabilities. Since all the sensed data is computed in the cloud, the computing power of the crowd is not fully utilized. The Internet of Things has led to billions of sensors being deployed, and as a result, a large amount of data is being generated. But it is not affordable for everyone to deal with the large amount of data that is being sensed, and it can be very costly at times to process this data.

In [55], the authors propose an energy efficient framework that deals with mobile crowd-sensing data in a distributed and on-demand fashion named CMOSDEN. This framework consists of a context-aware module along with sensing capabilities. Based on users' preference, the modules can be activated or deactivated, that results in saving of cost by reducing memory usage, CPU usage, or network usage. However, this framework was restricted to the users' context. In [56], the authors mentioned the problem of the existing centralized client-server model for mobile crowd-sensing, i.e., the server has to endure high operational costs, which results in poor scalability.

However, with the rise of computational capabilities and sensing capabilities in smartphones, a huge amount of data that is being sensed and generated can be well handled by peer-to-peer architecture. The framework can reduce the operational cost of centralized servers in a mobile crowd-sensing architecture. But in order to motivate the peers to participate, an incentive scheme needs to be implemented. Hence, they propose a generalized but best way to handle the market equilibrium incentive scheme to persuade the peers to share the sensed data.

In this framework [57], new communication methods are being enabled between the participating devices of the mobile crowd by taking into account the collected sensed data, decreasing the rate of data transfer and the operational cost of the cloud servers . The framework helps in keeping the sensed data in the device itself and then communicating and sharing the sensed data when needed by the other participants of the crowd directly from the device itself while keeping the identity of the device virtual and providing the crowd with real time data. This framework helps to build a virtual identity for the owner by accumulating information about the owner of the smartphones. The user is given the control over what types of services it wants to run on the device, thus helping to maintain user privacy by controlling the access mechanism of the device.

In [58], the authors address the issue of energy consumed by crowd-sensing applications. The authors note that in the case of indoor crowd-sensing, the devices collect both position data and sensing data. So, the authors propose server side localization techniques so that the smartphones can be located by analyzing their beacon exchanges with the wireless Access Points (APs) at the server side. Thus, only sensing data needs to be sent, thereby saving energy consumed for sending location data. This technique heavily depends on the infrastructure and may raise security concerns.

In [59], the authors propose a framework for data collection through sensing for utilization in various smart applications. The authors suggest that context awareness can reduce the sensing and communication costs. A multidimensional context model is put forward to record the related contextual information and then implement it in the MCS framework to reduce energy consumption and optimize task allocation costs. The goal of piggybacking crowd-sensing [14] is to reduce the cost of energy consumed. Data is piggybacked during a phone call or when the connected nodes exchange data with remote servers. This type of framework is suitable for delay tolerant MCS applications. Minimizing energy consumption through scheduling is discussed in a few works.

In [60], authors discuss one of the most important issues of balancing the amount of data sensed with energy consumption. The amount of energy consumed for the sensing and transmission of data can be reduced by scheduling the data. Both online and offline scenarios have been considered, where, in the case of the offline scheme, the entire task schedule is known beforehand and does not change with time. In the case of online scheme, two methods were proposed, namely, First-In-First-Out (FIFO) task model and the arbitrary deadline task model. In the case of online scenarios, the tasks are dynamically allocated without advance knowledge of the tasks. In [61], authors investigate the issue of scheduling various tasks of sensing that are assigned to smartphones with the aim of minimizing the energy consumed while sensing and maintaining the Quality of SenSing (QoSS). Here, two conditions are considered.

Firstly, the Minimum Energy Singlesensor task Scheduling (MESS) problem is considered, and the authors put forward a polynomial-time optimal algorithm to solve it. For the other scenario, a general case is considered where multiple sensors are used for sensing. The authors present an Integer Linear Programming (ILP) formulation as well as two effective polynomialtime heuristic algorithms for the corresponding Minimum Energy Multi-sensor task Scheduling (MEMS) problem.

ShareSens, an application, that merges opportunistically the independent sensing requirements of applications is proposed in [62]. This is done by introducing schedulers for the sensors. These schedulers help in determining the lowest rate of sensing that would satisfy the requests. Custom filters are then utilized to filter only the required data that is used for each application. Few research works are conducted where a user in the vicinity is selected for crowd-sense, depending on a number of factors. Thus, other devices in that area may save energy by not lending themselves to crowd-sensing.

In [63], authors propose a distributed framework for data sensing and collection in opportunistic MCS systems. The framework proposed by them is based on two policies namely, the Collector-friendly policy and the Smartphone friendly policy. The Collector-friendly policy is similar to continuous sensing, whereas the second policy minimizes the sensing and transmitting costs of the data by estimating the cost incurred for performing the operation in a distributed way. The proposed mechanism arrests the draining of participants' batteries completely for continuous usage or involving too little. The distributed nature of the framework enables the participants to determine the duration during which the device is going to participate in sensing. In [64], the authors propose a method that helps to select users for participatory sensing efficiently. It takes into consideration the remaining battery level of the participants and their willingness to participate while choosing the participant dynamically. Tasks are distributed in such a way that it reduces the probability of an individual not completing the task assigned to them.

In [65], the authors propose a framework that is energy efficient and cost-effective.

They classify users into two groups. In the first case, the users pay for the data they sense and send to the server. The aim in this category is to reduce the energy consumed while sensing the data. In the second case, the cost of sending the data that is being sensed is minimized by utilizing free communication technologies. However, energy and other resource consumption for crowd-sensing should also be minimized, even when utilizing free communication technologies, as smartphones have limited battery power. However, if they are connected to the power source, it may always participate in crowd-sensing.

Thus, framing an energy efficient strategy for crowd-sense is a dynamic problem that not only depends on what to sense [59], where to sense [58], or which device to use for sensing, as in [63], but also depends on when to participate in crowd-sensing depending on the condition of the device. Scheduling mechanisms [60] can be used to solve this problem, however, assessing the present condition of the device, such as its current load, and recharging probability, plays a key role here. MDP has been used in the literature to formulate battery management techniques that form a balance between different energy consuming tasks, as in [66]. [67] but not for crowd-sensing. Thus, the problem can be better formulated using MDP, which attempts to frame strategies to be adopted dynamically depending on various computational states and input probabilities. While offloading data or tasks to the cloud, often failure is encountered; hence, various types of failures need to be studied.

2.4 Offloading systems

The transfer of computation intensive tasks from smart handheld mobile devices (offloadee) to remote backend servers, which are resourceful servers (surrogates) for execution, is termed cyber foraging [68] or offloading [69]. When a task is offloaded, the load on the mobile devices is released, which leads to an enhancement in the performance of the mobile devices [70]. Generally, the tasks are offloaded to the cloud servers as it provides unlimited access to the resources at any point in time remotely [71]. Hence, mobile cloud computation offloading can be referred to as a platform of offloading in which the cloud servers act as surrogates and the offloadees are the mobile devices [72]. The surrogate domain should have certain features such as reliability, amalgamation of varied categories of services on different platforms, scalability, enhancement in the capacity of storage, cost reduction, longer battery life, etc., [73].

Load balancing in offloading systems by cloud computing is very different from the traditional migration model that is used in grid computing or multiprocessor systems [74], where in mobile offloading, the program or code is transferred to servers that are not immediately near the mobile device domain, however, in the case of grid computing, the program or code migration happens between servers in the same computing region, i.e., grid [75]. In recent times, the offloading of tasks has been shifted to edge devices that are near the vicinity of mobile devices, which reduces the overall computation time of the system. Hence, mobile-edge computing (MEC) refers to the migration of the computation intensive tasks to the servers, which are positioned at the edges of the networks in a fully distributed fashion. This architecture enhances the battery life of the mobile devices, reduces the overhead of communication and execution delays, and increases the computation capabilities of the mobile devices [76], [77], [78].

2.5 Classification of faults and failure in offloading systems

Offloading system failures can be classified into four kinds: 1) crash failure, 2) omission failure, 3) transient failure, and 4) security failure.

1. Crash failure: One of the reasons for crash failure is software aging [9]. Soft-

ware aging is a very familiar event where a software's performance deteriorates with time gradually, and at a certain stage of evaluation, it completely suspends functioning and the occurrence of crash failure happens. Software aging can happen due to corruption in data, unavailability of storage space, draining of the resources of the operating system, or numerical round-off error aggregation. There may occasionally be situations where, during the offloading of the tasks to the surrogate, the mobile device or the surrogate may stall due to a power cut or overloading of different tasks.

2. Omission failure: This type of failure may occur while offloading the data. Data from the mobile devices may not get transmitted to the surrogates, or the results from the surrogates may not get transmitted to the mobile devices, i.e., the request by the mobile device or the results from the surrogate may get lost during transmission, resulting in omission failure. There may be several factors for omission failure in offloading data for execution of the tasks to the surrogates, namely, long delays in wireless networks, and low bandwidth, which can lead to unreliable networks [9], [10]. In certain situations, the surrogates may also not be available to provide services to mobile users, which may also lead to omission failure [79].

3. Transient failure: Interruptions may also occur while offloading the data due to undetermined or hidden agents, which may temporarily halt offloading of the data. These types of faults often occur momentarily; hence, it is generally very difficult to detect such faults. The standard of wireless links through which transmission occurs is not very predictable, and due to the mobile nature of the smart devices, the connection between the surrogate and the mobile devices may often oscillate, which may result in unforeseen transient failures [80], and these types of failures are difficult to model.

4. Security failure: As the number of mobile devices participating in the offload-

ing increases, the access points also increase, resulting in a huge threat to the security of the data. Timing attacks [10], [81] result in certain security failures in offloading. An attacker can extricate important information maintained in the system by carefully scrutinizing the time taken to place various queries by the mobile devices, since these devices have the command over the resource rich surrogates over wireless networks for some duration of time. One way to protect the data and resources from being attacked is through virtualisation, where the data and the resources are isolated, though this can not prevent timing attacks. Encrypting and decryption of user data by the server master key is another way of securing the data while offloading [10] though this mechanism is also prone to attack by analyzing the response time of a certain service over a period of time by keeping a record of each response time of the service to the request.

The various categories of faults and the failures caused by them, in offloading systems that are discussed above are summed up by Fig. 2.5. The Fig. 2.5 also depicts the feasible tolerance methods.

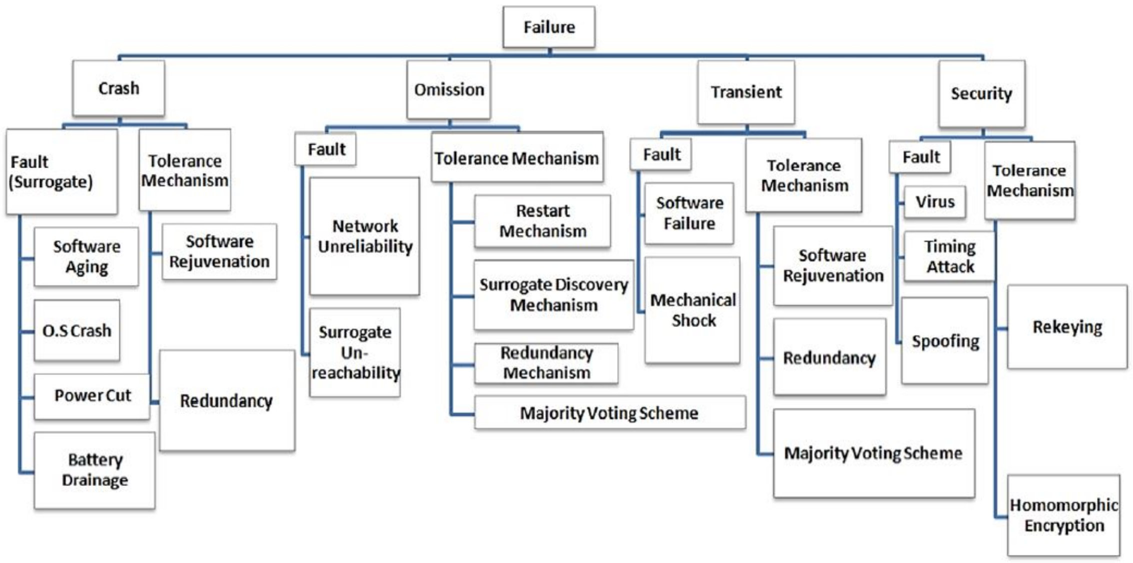


Figure 2.5: An overview of different faults with their existing and probable fault tolerance mechanisms

2.5.1 Different fault-tolerance approaches

This section gives a condensed idea of the various prevailing fault tolerance approaches for offloading systems. It has been observed that to obtain a fault tolerant offloading system, in various researches studies, crash failure or checkpointing is utilized [79], [69], [80]. In [79], snapshots of the execution of any application at the offloadee and surrogate point are taken at a fixed time interval. The mobile user keeps receiving this checkpoint data from the surrogate points. The latest data checkpoints of the application that was executing before the failure are loaded by the offloading system to recover the system from failure. In this way, the mobile devices can handle crash failures that may occur due to charge depletion in the battery or an abnormal shutdown. This method is also helpful in handling crash failure such as shutdown, omission failures, or wireless link failure at the surrogate side when the mobile device's mobility makes it difficult for the mobile device to reach the surrogates.

As energy is a scarce resource in mobile devices, the decision of whether to offload a task to the remote server or run it locally in the mobile device becomes very crucial and is decided based on the energy utilization for performing the tasks. Hence, in [82], the authors have segregated the tasks into offloadable and unofflaodable tasks. By duplicating the components of the tasks in the mobile device as well as the server, which is remotely in the cloud, fault tolerance can be obtained when there is a failure in the network. The tasks are switched between them, and once the network recovers from the failure, the task execution continues at the previous point of execution. This way, a seamless fault in the network is handled.

In [83], the authors propose a scheduling technique that is composed of implementing a genetic algorithm and DNA combination underneath the precedence level. This proposed method lessens the ratio of energy that is being consumed, minimizes the time needed for processing a task without missing the deadline of the tasks.

This method also bestows reliability by recovering the data that is processed in the mobile devices successfully and avoiding failure on virtual machines. Here, based on the deadline of the task, the offloadable tasks are sorted in ascending order. The virtual machines are allotted the smaller sub-tasks by breaking down the large offloadable tasks. The precedence level is implemented by utilizing genetic operators for task scheduling. A genetic algorithm is implemented to reschedule the subtask from the stored list, to other virtual machines, when there is a failure at a certain level to execute the subtask. Parallel processing at multiple levels helps in avoiding the failure of the sub-tasks.

In [38], the authors propose a technique that enables a mobile device to reduce its energy consumption by offloading tasks to remote servers without partitioning the tasks. The part of the program that consumes the most energy is offloaded to the cloud server. The checkpoints of a program are shared with both the mobile device and the remote server from time to time. When there is any network failure, the tasks are executed at the local site, i.e., mobile device, from the latest checkpoints. But in this approach, the program state does not stay consistent when the offloading is done from multiple and concurrent threads.

In [69], the author proposes an approach in handling omission failure. When there is a timeout, the system checks for errors. On finding a failure in the system, this approach sends the control back to the local proxy. At this point, the proxy follows one of the two tasks. It either locally restarts the process or it looks for another MAUI server and allocates the tasks to that server, which restarts the whole process afresh.

Omission failure can also be handled by COMET [84], when there is surrogate unreachability. Prior to making any permanent changes for any operation, all the remote data needs to be present; otherwise, the whole procedure reverts back if there is any loss of data due to the unreachability of the surrogate, before the procedure is finished. To recover from failure, all the threads need to be started locally by the offloadee. Hence, it is essential to store the information to continue the execution of the tasks in case of the unreachability of the surrogate. The surrogate's unreachability or loss can be determined if the surrogate does not respond quickly or if the surrogate connection is closed.

In order to curb aging [9], Software Rejuvenation [85]- [86] is suggested. It implies that when an application has aged, the application is terminated gracefully, and then it is once again started from a clean internal state. For the successful completion of an application execution, the application is rejuvenated periodically, after every interval of rejuvenation, by restarting it preemptively from its previous clean checkpoint. This reduces the chances of failure in the system. To eliminate the collected errors while running an application, the software is stopped intermittently before starting the software again. This procedure provides the stability of the surrogate and prevents the aging of software in the long run, though it may reduce the performance of the system for a short duration. Poor network condition can be cited [9] as the reason for degradation in the performance of the offloading systems, whereas for boosting the performance of the system conditions, restarting the system, a simple recovery technique, is often preferred [10], [87], which can reduce the network failure.

Due to the ever shifting nature of the offloadee, surrogate reachability often results in transient failures. In such cases, a discovery mechanism to locate a surrogate can be adopted. Locating the position of the surrogate can be obtained by CRoSS [88], in a wireless mobile environment, which will help in realizing the status of a surrogate, i.e., whether it is reachable or not [89], [79]. Often, in offloading systems, long downtown experience and network unreliability are accountable for the failures. In those limited cases, executing the tasks locally is often an alternate way [87], though not always preferable when the tasks are computation intensive. This method of local execution functions well when the tasks are small and not computation intensive and the performance is better when execution is done after waiting for an optimal amount of time for the network to recover. Security is often an issue in offloading systems. Timing attacks cannot be avoided by traditional mechanisms of security.In [10], the authors provide a solution by implementing a re-keying mechanism where the keys are frequently changed.

In [90], the authors utilizes the server master key for encryption and decryption of the user data. To further enhance the security of the system, it is proposed to change the master server key at a favourable time when the access of the user is low, such as during the night. In order to enforce security in offloading systems, procedures such as homomorphic and steganography are utilized [91]- [92], where homomorphic encryption does not need to decrypt the encrypted data but still provides security of data, integrity, as well as privacy, and in steganography, the original image is hidden under the disguise of a cover image so that the original image cannot be identified easily.

In [91], to protect the data, the image retrieval technique is utilized in homomorphic encryption. The authors in [92] utilize steganography to protect the original data, where the image retrieval technique is used. As can be observed, the need of the hour is a concise fault tolerant offloading approach since this system is prone to various types of failures, like omission or crash. Checkpointing is an efficient technique to deal with crash failures [79],[82]. In [80], the proposed approach picks out two surrogates, primary and secondary, where the checkpoint is stored at the surrogate side for better efficiency. But this procedure is not very suitable for transient or omission failure. But this approach does not compare the results with some benchmark applications, which is used to determine the efficiency of the system.

2.6 Various task allocation approaches to the edge servers

In [93], the authors utilized an approach based on fuzzy logic to handle the workload orchestration. Workload orchestration can be described as a process of deciding where and how a task should be computed (edges or cloud) in an n-tier architecture. The authors proposed to capture the instinct of a real-world administrator for automating the management system that performs work orchestration. The authors have not considered task migration between edges and clouds. In [94], the authors presented a collaborative scheme of task offloading built on FL in mobile edge computing (MEC), which is distributed in small, concentrated networks. The authors have not considered the mobility of the users, which restricts the usability of mobile devices.

In [95], the authors proposed a concept of edge infrastructure providers (EIPs) model and termed it Edge federation, where edge and cloud are both part of the federation so that the latency critical tasks can be serviced by resource cooperation. The authors have characterized this resource sharing process as a LPP (Linear Programming Problem) and transformed it into solvable form using a dynamic algorithm. The authors have considered a few fixed-length time slots and have not considered dynamic prediction of the length of time slots.

In [96], the authors have considered a 3-tier architecture, distinctly differentiating the cloud-fog-edge layers, and the fog layer is considered to be more resource rich than the edge layer for Vehicular Ad hoc Networks (VANET). A resource management approach was proposed utilizing integrated fuzzy logic that determined the fittest vehicle for task offloading. The authors in [97] explored the issue of task scheduling in vehicular edge computing (VEC), and these tasks could be sent off to the roadside units (RSU). The aim was to diminish the average time for completing a task. Here, the authors have not considered the communication delay of the tasks and the simulation have been done on small data set. There have been a lot of research initiatives in multi-clouds [98], [99], [100] or edge clouds [101], [102] for scheduling services to decrease energy usage.

Here, in [103], the authors have proposed a mathematical task offloading model for MEC- IoV utilizing PSO techniques, for offloading the task to the cloud edge computing system. Here, the authors have not considered the various resource constraints, such as QoS, CPU requirements, or the priority of the tasks. In [104], the authors have proposed a task offloading strategy to the edge servers based on PSO. But there are certain issues with controlling the parameters of PSO, and the algorithm has a reduced convergence rate while running the iterative process. Also, the authors have not taken the reliability of the completion of the tasks into account.

In this work [105], reinforcement learning, Q-learning, is utilized for taking the decision of task offloading to the edge servers in industrial IoT, where resource allocation problems and task offloading are visualized as a sum of cost delay problems. But the authors have not compared its work with some other benchmark algorithm to understand its full potential. Also, in a noisy environment, it is found that Q-learning slows down its learning process due to the overestimation of the action values. Here [106], an approach is proposed that reduces the overall energy consumption for IoT mobile devices. Though the authors have claimed that the proposed approach has significantly reduced the energy consumption of mobile devices, however, they have not compared it with any standard benchmark methods to validate their theory. In [107], the authors propose a strategy for offloading the tasks to edge servers, considering a few key parameters such as task size, computing requirements etc. The task scheduling on the edge servers is based on an improved auction algorithm.

In [108], the authors proposed an offloading architecture based on device-to-device (D2D) for energy efficient communication for MEC, where transmission is done by utilizing the cooperative relay technique. The author also proposed technologies for offloading and load balancing on the edge servers. But the techniques were not compared with any existing benchmark techniques to understand their efficacy. Here, [109], the authors proposed a load balancing method for tasks in a cloud environment utilizing PSO, where the tasks are offloaded to virtual machines located in the cloud from the overloaded virtual machines. However, in this work, there are no comparisons made with benchmark techniques to understand the proposed methods effectiveness.

2.7 Summary

This literature survey gives an idea of the various topics that are being researched related to this thesis. A brief idea about the importance of mobile crowd-sensing and sourcing and a comparative study is discussed here. In this existing literature study, it was found that mobile crowd-sensing and utilizing crowd intelligence were done in a separate manner. But integrating the two paradigms is revolutionary in the sense that it can be applied to various new fields of application. It not only enriches a mobile device that is not aware of its surroundings, but it also helps the mobile device contribute to the process of crowd computation after the device has acquired enough knowledge of the surroundings and is in a position to share information without the use of remote servers.

Utilising both paradigms, i.e., mobile crowd-sensing and crowd computing, the positive outcome lies in the fact that the local crowd participates, senses the useful information of the neighbourhoods, shares the information with the surroundings, and receives rewards for doing so. This kind of applications, could be local in nature, so there is no need to send and compute over the Internet, however, the problem can be solved by the crowd in the vicinity themselves. The various types of mobile data sensing and collection frameworks used by mobile crowd-sensing applications, which focus on reducing the energy that is consumed, are also explored in this chapter.

Offloading systems and the various categories of faults and failures are analyzed, along with different fault tolerant approaches. This chapter also explores the various task allocation approaches in the edge servers. Detail studies and exploration of the topics will be discussed in the next few chapters. This has motivated us to design a framework which integrates these two paradigms i.e., mobile crowd-sensing and mobile crowd sourcing, and the details of the work is explained in the next chapter.

List of Publications:

Journal:

1. **Ray, A.**, Chowdhury, C., Bhattacharya, S. and Roy, S., 2023. A survey of mobile crowd sensing and crowdsourcing strategies for smart mobile device users. CCF Transactions on Pervasive Computing and Interaction, 5(1), pp.98-123.

Chapter 3

Framework for Mobile Crowd-sensing

In this chapter, an efficient framework is proposed which integrates the two paradigms, i.e., mobile crowd-sensing and crowd computing. This in turn has helped the mobile devices to process the sensed data locally in the device itself or can be processed by taking the aid of human intelligence of the crowd. So, the framework has enabled to handle real time response in short duration as the data is not getting transferred to the backend cloud servers for processing. In this manner, a reliable solution is provided to the user of mobile devices where energy utilization is lesser than when the data is sent to cloud for processing. Of course a certain amount of incentive is given to the service provider (in this case, the crowd who has participated in solving the task). The results that are acquired can be collected both by online procedure when there is internet connectivity or by offline procedure, i.e., with the aid of human intelligence. This not only reduces the transmission delay but also manages to efficiently utilize the energy of the mobile devices. Thus, this chapter discusses the issues related to energy utilization of the mobile devices, objectives and contributions as follows.

Problem description Efficient utilization of the energy of the mobile devices which leverage the crowd for real-time data processing, saving energy and reducing reliance on the cloud.

Objective The objective of this framework is to compute a given task locally by obtaining the information from the crowd that is sensed by the mobile device without the aid of cloud servers, i.e., offloading the task to the cloud. For example route recommendation, obtaining location information, image comparison and identification etc.

Contribution The contribution in this chapter is that an efficient framework is proposed which combines the two paradigms, of mobile sensing and crowd computing and functioning as a unit to process the data locally without the aid of backend cloud server and providing a reliable solution to the mobile user within a short span of time for a certain amount of incentive given to the service provider.

The rest of this chapter is organized as follows: Proposed approach is discussed in Section 3.1 followed by details of functioning of the framework explained in Section 3.2. Section 3.3 discussed in details about the implementation of the framework by online and offline procedure, Section 3.4 explains in details the experimental setup for implementing this framework, followed by results and discussion in Section 3.5. Finally, the Section 3.6 summarizes the chapter.

3.1 Proposed approach: designing of the efficient framework

An integrated framework for mobile sensing and crowd computing is needed for better realization of the task in a short span of time with limited resources. For example, a person lost a bag in the airport and wants to find the bag. This can be achieved by the proposed framework where the initiator, i.e., the person who lost the bag, broadcasts the picture of the bag to the local crowd in the airport. The contributors process the image at their mobile device and try to find the location of the missing bag through the various images captured nearby. The crowd then sends back the images it has processed. The initiator then retrieves the location of the missing bag from the various images it has received from the contributors of the crowd. In this case, the total work is done with the help of crowd without sending the data over to remote server for analyzing the tasks saving a considerable amount of time.

In the proposed work, N number of smartphones are connected to one other by BLE (Bluetooth Low Energy) where few of them have connectivity to remote servers through the Internet. Usage of BLE technology for the proposed work is useful as BLE is infrastructure less technology, low powered, small in size and is inexpensive. They work well with low energy applications. The range of these devices is 330ft [110] which will help to solve local issues within the neighborhood. If the problem is outside this range then multi-hopping scheme can be utilized to solve the problem using the opportunistic network. A user upon receiving a query that is being broadcasted can execute the query within the device itself or it can forward the query to its neighboring smart devices and thus spanning the network topology beyond one hop. The number of hops would be decided by the requirements of the application. At first, when a query is broadcasted over the network, $k(< N)$ number of neighbors would receive the query but it may happen that not all of them may participate to perform the task. Among them, $k'(<= k)$ devices would respond to the query depending upon the eligibility criteria. A device may become a contributor in the crowd if it has sufficient energy to perform the task (more than 20% battery power) and the device is having no or minimum load at that point of time. Another factor that should be taken into account is the devices willingness to participate. If it is happy with the amount of incentive it receives for performing the task then the device may contribute. The incentive can be monetary or non-monetary [111]. A monetary incentive may be in the form of real money or virtual cash which the user can utilize in the market. A non-monetary incentive can be social, entertainment or service incentive. In case of social incentive a persons participation is based on the mental satisfaction for participating in the crowd sensing whereas entertainment incentive is where the participant derives entertainment out of providing the service, and the service incentive is that type of reward where the service provider of the mobile user provide a certain amount of free service for participating in sensing.

There are five phases in the framework from requesting help to disseminating incentive for getting the service.

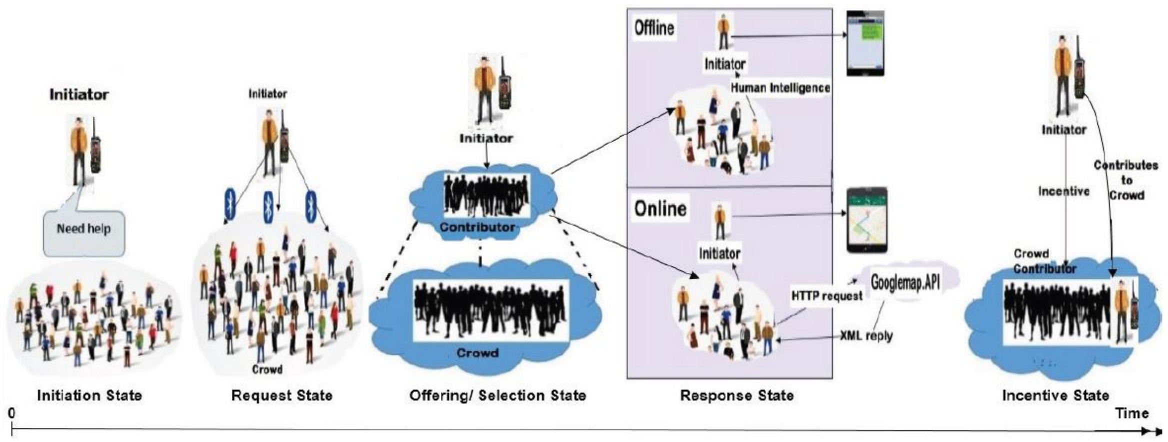


Figure 3.1: State transition diagram of the proposed framework

Different phases of the proposed framework has been summarized in Fig. 3.1. In the initial phase, the initiator who wants service from the crowd asks for help to the crowd. In the request phase, the initiator, who wants service, advertises its need to the participants of the crowd through BLE communication. In the offering or selection phase, a subgroup of the participant who are eligible for sharing or helping the initiator, called the contributor takes part in the crowd sensing. If the application needs to broadcast a piece of information to selected group it uses selection else the initiator broadcasts to the whole crowd. The contributors upon receiving request check its status and eligibility and accordingly replies back to the query in the response phase. The reply that is being sent by the crowd can be obtained online or offline. If it is through online then the contributors receive xml response through internet by unicast or when offline then it can happen that the human intelligence is being used to answer back the query. The advantage of offline mode is that it does not need infrastructure and the reply to the query can be sent through BLE technology with less power consumption. In case of offline mode, a mobile user may have the knowledge of the query hence replies back with the answer. The initiator after receiving the data from various contributor aggregates the data to obtain the total information. The initiator then disseminates the rewards to the devices from which it receives the requisite services in the incentive phase. The initiator now gaining the knowledge can contribute to the crowd for any further query thus enriching the quality of the contributors. A minimum number of participants must always be available in order to guarantee the reliability of the data.

Here, incentives are given with respect to the sensing by the mobile device but it does not talk about the incentive that can also be bestowed when a device is crowd computing at the same time. So, the methods of incentive that is given for sensing can be extended for computing as well.

3.2 Functioning of the framework: case study

In this chapter, the framework is implemented and the effectiveness of it is shown with respect to a route finding application. The initiator starts off with requesting for help to the crowd for route recommendation as depicted by initiation state of Fig. 3.1. This help can be achieved online, when the contributors are connected to the internet or offline when human intelligence is used. The application requests the initiator to mention the source, destination and the mode of transport the initiator would like to avail in order to find the route. Upon receiving the request the contributor performs the tasks or forwards the task to its neighboring devices. In order to evaluate the feasibility of the defined MCS platform, it has been applied in two different scenarios.

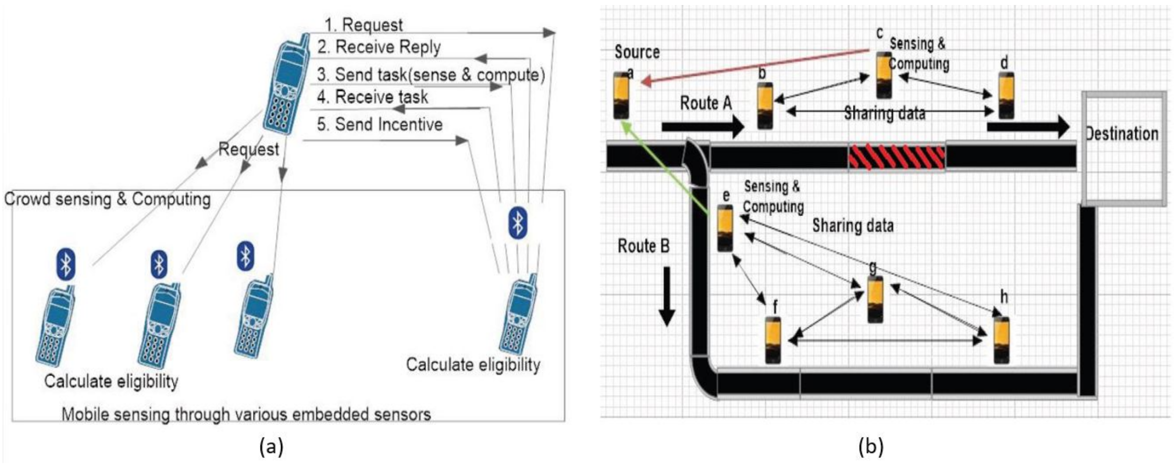


Figure 3.2: (a) Crowd computation done by participating crowd (b) Finding fittest route by help of crowd computation

- Case 1: Consider a scenario, in Fig. 3.2 (a) where a mobile user is having very low battery charge and the user is new to a location and wants to know the route to a certain destination. Obtaining location information through GPS would prove to drain the battery at a much faster rate. Hence, the user takes the help of the crowd to complete the task. Here, the user offloads the task of sensing and computing to nearby mobile devices via BLE which would require less energy. The offloading mobile devices after receiving the request calculates its eligibility to perform the task, i.e., finding the route to the destination. If the device is eligible it then computes the tasks and replies back. The initiating device then sends an incentive to the device which has completed the task.

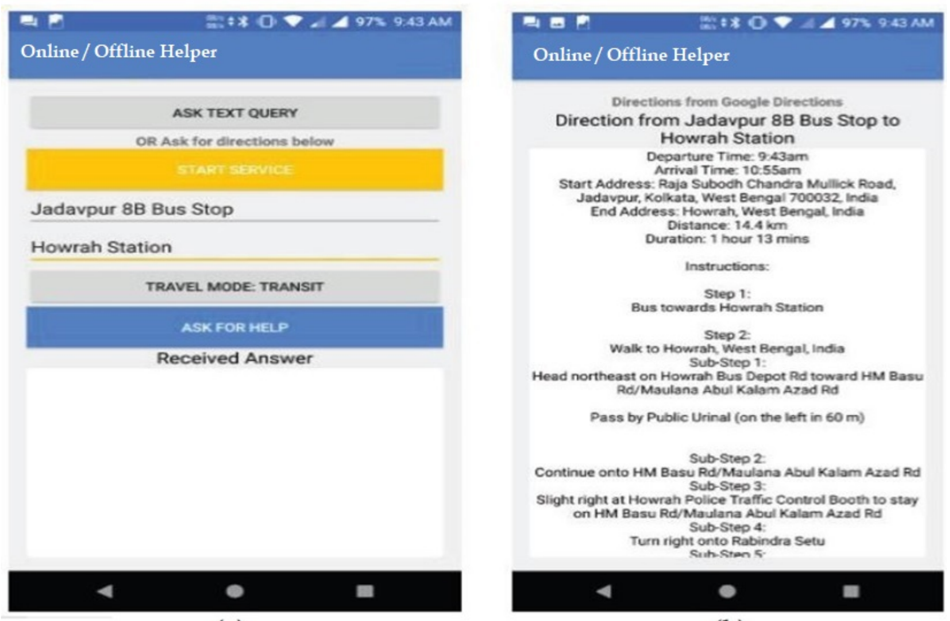
- Case 2: Another situation can be taken into account where an enquiry of the shortest route as well as the condition of the road is done by a user who is not aware of the route condition to the destination and new to the place. In the Fig. 3.2 (b), it can be seen that the GPS shows the shortest route to the destination, i.e., route A, but it fails to give the information regarding the condition of the road if it is damaged or blocked. As can be seen route A has a portion of it damaged and this information can be obtained from the crowd located at that place (in this case device c). The source node comes to know about the road blockage from user c. Thus, the source node could find that the best possible route in this scenario is route B, at that given point of time. Hence, though the shortest route to destination shown by GPS is route A but owing to the bad condition of the road, the fittest route to the destination is route B and this information is obtained with the help of crowd computation.

3.3 Implementation of the framework through online and offline procedure

An Android application is built for the proposed work which works in two modes, online when the contributors are connected via the Internet to the remote server and offline when there is no connectivity with the Internet. The device which needs help runs this application in its foreground specifying the parameters by HTTP request. The contributors who are there and wants to render their services have this application running in the background and they respond only upon receiving a notification from broadcasting. They respond online by xml reply after obtaining the result from Googlemap.api [1] over the Internet. Two or more mobile devices need to participate in order to run the application. When devices are participating in the application it denotes that the devices either want to help others or they need help. The device which needs help is denoted as the initiator and does not have the Internet connection but the devices which are willing to help are the crowd participants of the crowd who may or may not have the Internet connection to the device. The initiator broadcasts its need to the crowd through BLE communication. In this application, the initiator enters the source, destination and traveling mode and sends this information as a broadcast message over the network. The participants of the crowd whose devices have started the service have this application running

[1]https://developers.google.com/apis-explorer/

in the background and a notification of the broadcast message is received by the crowd. The contributors can either help the initiator by the offline procedure or by the online procedure. In the case of online help, the direction guidance will be fetched from the internet by the contributors, upon receiving the direction of the route is sent to the initiator via the BLE connection. In case of offline help, the participating crowd use human intelligence and send back the reply using some text message via BLE. The reply is sent through the unicast message to the initiator.



Jadavpur 8B Bus Stop
Howrah Station

Directions from Google Directions
Direction from Jadavpur 8B Bus Stop to Howrah Station
Departure Time: 9:43am
Arrival Time: 10:55am
Start Address: Raja Subodh Chandra Mullick Road, Jadavpur, Kolkata, West Bengal 700032, India
End Address: Howrah, West Bengal, India
Distance: 14.4 km
Duration: 1 hour 13 mins

Instructions:

Step 1:
Bus towards Howrah Station

Step 2:
Walk to Howrah, West Bengal, India
Sub-Step 1:
Head northeast on Howrah Bus Depot Rd toward HM Basu Rd/Maulana Abul Kalam Azad Rd

Pass by Public Urinal (on the left in 60 m)

Sub-Step 2:
Continue onto HM Basu Rd/Maulana Abul Kalam Azad Rd
Sub-Step 3:
Slight right at Howrah Police Traffic Control Booth to stay on HM Basu Rd/Maulana Abul Kalam Azad Rd
Sub-Step 4:
Turn right onto Rabindra Setu
Sub-Step 5:

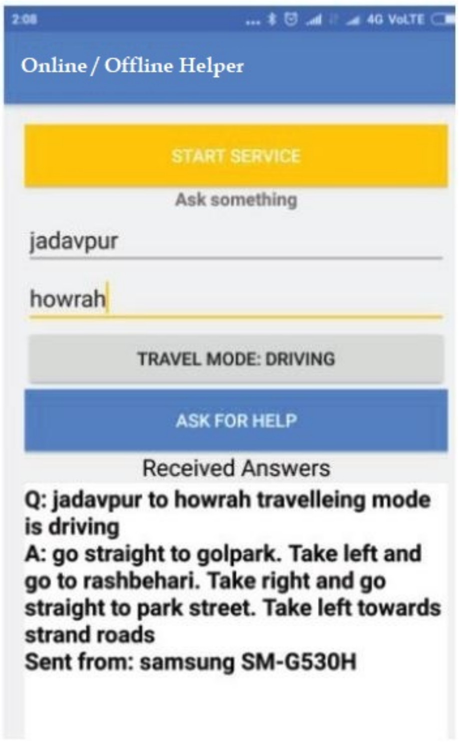
Figure 3.3: Snapshot of the application while getting direction online from the crowd

3.4 Experimental setup

Four different smartphone configurations are used for experimentation: Moto G4 plus with the battery capacity of 3050 mAh having 3GB RAM and using version 7.1.2 version of Android, Redmi note 4, 4100 mAh battery capacity and 4GB RAM using Android 7.0 version, Samsung Galaxy Tab E with 5000 mAh battery capacity and 1500MB RAM using Android 4.4.4 version and Samsung Grand Prime having Android version 5.0.2, 2500mAh battery capacity and 1GB RAM. The application is executed to obtain the route as well as the amount of battery that is being consumed and the amount of load on the CPU for running the task.

3.5 Results and discussions

An contributor could reply to a query in online/offline mode. Fig. 3.3 (a), shows the screenshot of the query done by the initiator and Fig. 3.3 (b), is the reply to the query by the contributor about the route to follow from source to destination during the online procedure.



Q: jadavpur to howrah travelleing mode is driving
A: go straight to golpark. Take left and go to rashbehari. Take right and go straight to park street. Take left towards strand roads
Sent from: samsung SM-G530H

Figure 3.4: Snapshot of the application while getting direction offline with aid of human intelligence

Fig. 3.4 shows the screenshot of the initiator after it has received its reply from the contributor. The initiator has specified the source and the destination along with the mode of travelling. The reply to the query is received via BLE to the initiator along with the specified route. For replying back the message to the initiator, the user id of the initiator is stored. Bridgefy Sdk [2] is used for this purpose. It is utilized for Android and iOS and also allows message passing between the two OS. Google provides an API that returns the direction details of two places as a json or xml response. The source, destination and traveling mode needs to be provided as HTTP request and in response xml file is sent. The xml response is then parsed and displayed as a text in the scroll-able text view area in the participants device.

The battery consumption during the offline technique is significantly very less of about 1.2W and the CPU load is only about 12.83% on the device. During the online technique the battery usage is almost the same, i.e., 1.16W as during online procedure. The Google route map was already downloaded in the device hence the cost of computation is significantly reduced. The CPU load in this case is 12.14%.

3.6 Summary

Mobile sensing and crowd computing is the new paradigm which utilizes the crowd computation and social interactions with the help of mobile devices to obtain a distributed computation in large scale. It is known fact that better performance can be obtained through the help of community or crowd computation. In this work, a realistic framework is furnished for mobile sensing and crowd computation. But in order to motivate the crowd it is necessary to satisfy and keep the crowd happy. This is influenced by resource consumption by the mobile device and the rewards it receives on performing a task as a participant of the crowd. Crowd sensing and crowd computing not only helps a device to gain information about surroundings but also later this device can be used as another resource and can participate in sharing knowledge with the crowd. This work combines these two paradigms to provide a framework for better task realisation without the utilization of the backend servers. However, when the data is sent over the cloud, many issues come into the picture.

[2]:https://www.bridgefy.me/developers.php

In order to enable these handheld devices to participate in mobile crowd-sensing and crowd computing, these devices require to be active for longer duration of time. So, the devices need to take the decision of when to crowd-sense and when not to, based on certain criteria. The next chapter discusses in details about this strategy.

List of Publications:

Conference:

1. **Ray, A.**, Mallick, S., Mondal, S., Paul, S., Chowdhury, C. and Roy, S., 2018, December. A framework for mobile crowd sensing and computing based systems. In 2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS) (pp. 1-6). IEEE.

Chapter 4

Strategic Decision for Mobile Crowd-sensing Using MDP

In mobile crowd-sensing, smartphone users take part in sensing and then share the data to the server (cloud) and get an incentive. These data can be utilized for providing better services to improve quality of life. Batteries used in smartphones constrain the usability of these devices for longer charge cycles. Hence, maintaining a balance between energy consumption due to crowd-sensing application and that due to the current computational load on the device is the need of the hour. Hence, for keeping the device participating in the mobile-sensing an energy efficient strategy is needed which will motivate the crowd to participate and share the information. Thus, this chapter discusses the issues related to energy efficient strategies from the perspective of end users, i.e., mobile devices, objectives and contributions as follows.

Problem description Mobile crowd-sensing (MCS) has become a promising paradigm for cross-space and large scale sensing. The issue is to find how frequently a smartphone would lend itself for crowd-sensing while taking into consideration various conditions of the device.

Objectives This chapter proposes an energy efficient strategies where a smart handheld device takes the decision of participation in crowd sensing considering possible device conditions depending on various factors namely, remaining power of the device, the load on the device at that point of time, battery recharging and the crowd-sensing incentive. The problem is formulated using MDP which attempts to frame strategies to be adopted dynamically depending on various computational states and input probabilities. Such a formulation is solved prior to deployment to find out the most suitable crowd-sensing strategies. Then, such strategies, as obtained by solving MDP formulation, are implemented to work in real-time. The performance of MDP based strategic sensing is compared with other state-of-the-art strategies such as random or continuous sensing crowd-sensed indoor localization is considered as the MCS application here.

Contributions The main contributions of this chapter are:

1. A novel strategy is proposed which aids the mobile device to decide when to participate in crowd-sensing and it is formulated using MDP to devise energy efficient strategies. These strategies are used to derive trade-off between charge cycle of smartphones and participation in crowd sensing.

2. The MDP based strategies are implemented in an Android application to facilitate crowd sensing in real time.

The remainder of the chapter is organised as follows: Section 4.1 discusses in details the MDP for the system model. In Section 4.2 proposed approach is discussed. Section 4.3 discusses in details the designing of the crowd-sense application in indoor localisation and Section 4.4 explains the implementation of the approach. In Section 4.5, experimental setup is explained in details and Section 4.6 discusses the results and discussions followed by the summary in Section 4.7.

4.1 System model using Markov Decision Process

The system model is based on a two-tier server-client architecture where smartphone applications on the client side are connected to the MCS server through the Internet. The smartphones periodically sense, aggregate and send collected data to the MCS server. The problem addressed here is that, how frequently a smartphone would lend itself for crowd-sensing while striking a balance between maximizing incentive and minimizing the cost of energy consumption by the device due to crowd-sensing task.

To formulate the problem MDP is applied. MDP is a discrete-time state transition stochastic process which provides a mathematical framework for making any reasonable and rational decision when the outcome is partly random and partly controlled by the decision makers [112]. A Markov Decision Process is represented as a five-tuple: (X,J,P,R,γ). The notations are explained in Table 1. MDP is used to find a sequence of actions such that the resulting sequence of states maximizes the total discounted reward. MDP is a model that is solved through dynamic programming like value iteration to find a rational solution. Discounted reward signifies that reward obtained in a possible future state is considered while taking a decision in the current state.

An optimal policy π^* needs to be found, which maximizes the expected utility (calculated based on reward) of each state, i.e., given any policy π and any other x, $\pi^*(x) \geq \pi(x)$. So, the criterion for finding a good policy is that, for each state x, the policy should maximize the expected reward R in that state if that policy is executed. Value iteration is an iterative procedure which calculates the expected utility (value function) of each state, using utilities (value) of previous/neighboring states until two successive utilities are close enough, i.e., $max.x_i|U(x_i) - U'(x_i)| < \epsilon$, where ϵ is a threshold value (smaller the value better is the algorithm). That means the algorithm converges when the maximum difference between two successive value

Table 4.1: Notation used in MDP

Symbol	Description
X	A finite set of states
J	A finite set of actions (j_t) to be taken
P	Transition probability matrix, where $p(x_t + 1\|x_t, j_t) = p(x_{t+1}\|x_0......x_t,$ $j_0.....j_t)$ describes the state transitions. It plays the role of the next-state function in a problem solving search, except that every state ($x_t + 1$) is thought to be a possible consequence of taking an action (j_t) in a state (x_t). So it can be represented as a set of square matrices one for each action, indexed in both dimensions by states. The sum of all transition probabilities arising from a state is 1.
R	Reward matrix where $r(x_t, j_t)$ is the reward function that calculates the immediate reward (or expected immediate reward) received for state transition from x_t to x_{t+1}.
γ [0,1]	Discounting Function represents the importance of future reward in present reward.
$\pi(x)$	One of the policy where , $\pi^*(x)$ is optimal policy
U (x)	Expected resultant value at each state

functions is less than ϵ. Given the utility matrix, a policy can be chosen according to the maximum expected utility, i.e., $\pi^*(x_i) = argmax_j \sum_k P^j_{ik} U(x_k)$ [113]. Hence, if the maximum utility (value) is known for a state, the optimal policy can be found. This procedure has a finite number of steps (k) and finite number of actions has been taken under consideration. This reduces the computational complexity. Thus, a policy needs to be found, for every initial state x_0 which will result in the maximal expected sum of rewards from times 0 to (k) [1].

4.2 Proposed approach implementing MDP

MDP formulation for the MCS context is discussed in details in this chapter. The system model considers a two tier architecture where smartphones are connected to the MCS server through the Internet. The server, at times, asks the smartphones to sense and share data and gives an incentive for that work. The problem addressed

[1]https://ocw.mit.edu/courses/electrical-engineering-andcomputer- science/6-825-techniques-in-artificial-intelligencesma- 5504-fall-2002/lecture-notes/Lecture20FinalPart1.pdf.

here is that, how frequently a smartphone would lend itself for crowd-sensing while striking a balance between energy and incentive. Remaining energy of a smartphone plays a key role in this strategic decision. Energy of a smartphone may depend on its remaining energy, current operational load and possibility of recharging. So for MDP formulation, the state (X_t) of a smartphone at a time instant is considered in terms of its remaining energy (L_t), current operating conditions (A_t) and recharging probability (Y_t). A decision making agent observes the state and then takes decision of choosing some action from the set of given actions, J_t (0 not to crowd-sense and 1 to crowd-sense). Thus, the system at time t can be denoted as:

$$X_t = (L_t, A_t, Y_t) \tag{4.1}$$

where

- $L_t \in \{0, 1, 2, 3, ...N\}$ represents states corresponding to the energy available in the device at time t, where $l_t^{'}$ represents the amount of remaining energy at time t,
- $A_t \in \{0, 1\}$ where 1 represents a system with critical and power hungry applications running and 0 represents a system that is lightly loaded and
- $Y_t \in \{0, 1\}$ represents recharging status of the device. Y_t is 0 when the device is not connected to a battery source and is 1 otherwise.

Action performed at state X_t is given by $J_t \in \{0, 1\}$. A discrete time model is considered with time slotted in intervals of unit length. For instance, operational load of a node is considered to remain same in one slot. This assumption is realistic as time division multiplexing is used in smartphone operating systems.

Current operational load of smartphone is described by a correlated, two state process. If the system is heavily loaded in the current time slot, probability that the system remain heavily loaded (respectively, not loaded) in the next time slot is given by p_{on} (respectively, $1 - p_{on}$) where $0.5 < p_{on} < 1$ [114]. If the system is not loaded in the current slot, the operational load would remain low (respectively, not remain low) in the next slot with probability p_{off} (respectively, $1 - p_{off}$) where $0.5 < p_{off} < 1$ [114].

A smartphone can be recharged several times a day, even on the move through portable power banks. The energy generation process of a smartphone is modeled as a two-step process. If a smartphone is recharging in the current time slot, then the probability that it is recharging (respectively, not recharging) in the next time slot is q_{on} (respectively, $1 - q_{on}$) with $0.5 < q_{on} < 1$. In a similar manner, q_{off} (respectively, $1 - q_{off}$) symbolizes the probability of a smartphone not getting recharged (respectively, getting recharged) in the next time slot given that the smartphone is not recharged in the current time slot where $0.5 < q_{off} < 1$. The probability that there are i continuous time slots for recharging is $P[N = i] = (q_{on})^{i-1}.(1 - q_{on})$. Thus, the average length of a period of continuous recharge is:

$$E[N] = \sum_{i=1}^{\infty} i.(q_{on})^{i-1}.(1 - q_{on}) = \frac{1}{1 - q_{on}} \tag{4.2}$$

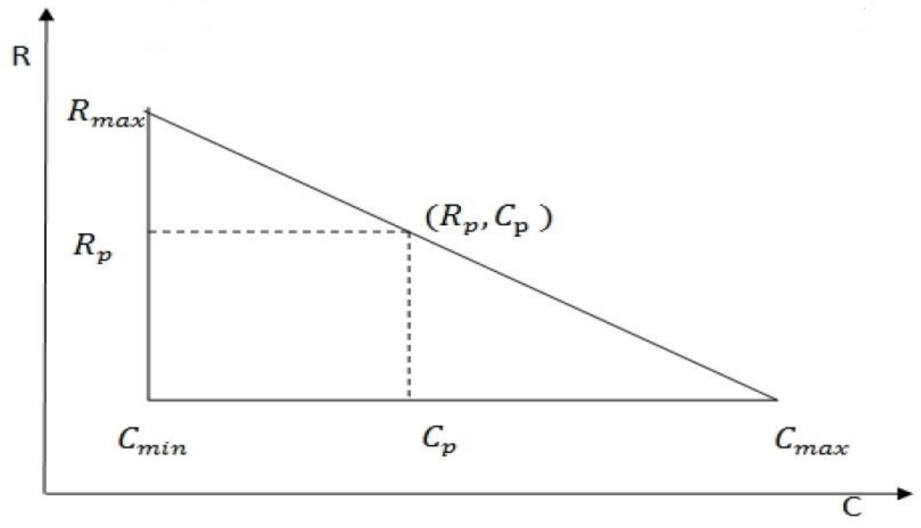


Figure 4.1: Calculation of reward w.r.t cost of crowd-sensing

Given these input parameters, Fig. 4.1 shows the variation of crowd-sensing cost (C_p) with reward (R_p). From the figure, it is evident that the following condition holds where $C_p \in [C_{max}, C_{min}]$ and $R_p \in [R_{max}, R_{min}]$. In this context, $R_{min}, C_{min} = 0$ (if the decision is no crowd-sense) and $R_{max} \approx C_{max}$ is considered here. C_{max} varies with current load and recharging options and so the formulation of reward $r(X_t, J_t)$ is as follows:

$$\begin{aligned} r(X_t, J_t) &= \rho_i - \frac{p_{on}}{q_{on}} \qquad if\ a_t = 1, y_t = 1, j_t = 1 \\ &= \rho_i - \frac{1-p_{off}}{1-q_{off}} \quad if\ a_t = 0, y_t = 0, j_t = 1 \\ &= \rho_i \qquad\qquad if\ a_t = 0, y_t = 1, j_t = 1 \\ &= 0 \end{aligned} \tag{4.3}$$

ρ_i is the constant reward for crowd-sensing. When the system is heavily loaded, even if it is being recharged, sufficient energy gets drained out too due to operational load and crowdsensing. This is reflected in the first condition of of Eq. 4.3. If the system is lightly loaded, even if it is not getting recharged, still it may crowd-sense depending on its remaining energy. However, some energy gets drained out due to this decision which should be included in the reward calculation. This is reflected in the second condition of Eq. 4.3. Besides, if the system is recharging and is lightly loaded, negligible energy gets drained out due to crowd-sensing. So here no negative reward is associated with this decision but gain the whole incentive as reward as presented by the third condition of Eq. 4.3. In all other cases, reward is calculated as 0. Given the device is in state X_t, the next state X_{t+1}= $(L_{t+1}, A_{t+1}, Y_{t+1})$ may be reached by taking action J_t in order to maximize $r(X_t, J_t)$.

$$\begin{aligned} L_{t+1} &= L_t + 1 \quad if \quad (L_t+1).\Delta l_t^{'} \ \leq \ l_{t+1}^{'} < (L_t+2).\Delta l_t^{'} \\ &= L_t - 1 \quad if \quad (L_t-1).\Delta l_t^{'} \ \leq \ l_{t+1}^{'} < L_t.\Delta l_t^{'} \\ &= L_t \end{aligned} \tag{4.4}$$

where $\Delta l_t^{'}$ represents the energy difference corresponding to successive energy levels of L_t and the remaining energy at time $t+1$, is denoted by $l_{t+1}^{'}$. N energy levels of L_t are set in a way that, in one slot, the maximum energy gained or lost does not exceed $l_t^{'}$. The $l_{t+1}^{'}$ is given as follows:

$$l_{t+1}^{'} = l_t^{'} + g_t - l_t \tag{4.5}$$

where g_t is the amount of charge gained due to recharging and l_t is the amount of charge lost while crowd-sensing as follows:

$$\begin{aligned} g_t &= c \quad w.p. \quad Y_t q_{on} + (1-Y_t)(1-q_{off}) \\ &= 0 \quad otherwise \end{aligned} \tag{4.6}$$

System load may vary from one state to the next state depending on the following formula:

$$\begin{aligned} A_{t+1} &= 1 \quad w.p. \quad A_t p_{on} + (1-A_t)(1-p_{off}) \\ &= 0 \quad otherwise \end{aligned} \tag{4.7}$$

The device may be recharged in the next state according to the formula:

$$\begin{aligned} Y_{t+1} &= 1 \quad w.p. \quad Y_t q_{on} + (1-Y_t)(1-q_{off}) \\ &= 0 \quad otherwise \end{aligned} \tag{4.8}$$

The amount of energy consumed due to crowd-sensing is denoted as follows:

$$l_t = \begin{cases} e_0 + e_2 + e_{cr} \; w.p \; [A_t p_{on} + (1 - A_t)(1 - p_{off})] \\ (1 - I_0(J_t))\text{where } A_t = 1 \text{ and } l_t^{'} > e_0 + e_2 + e_{cr} \\ \\ e_0 + e_1 + e_{cr} \; w.p \; [(1 - A_t)p_{off} + A_t(1 - p_{on})] \\ (1 - I_0(J_t))\text{where } A_t = 0 \text{ and } l_t^{'} > e_0 + e_1 + e_{cr} \\ \\ e_0 + e_2 \; w.p \; [A_t p_{on} + (1 - A_t)(1 - p_{off})] \\ (I_0(J_t))\text{where } A_t = 1 \text{ and } l_t^{'} > e_0 + e_2 \\ \\ e_0 + e_1 \; w.p \; [(1 - A_t)p_{off} + A_t(1 - p_{on})] \\ (I_0(J_t))\text{where } A_t = 0 \text{ and } l_t^{'} > e_0 + e_1 \\ \\ e_0 \text{ otherwise} \end{cases} \tag{4.9}$$

Here, $I_0(J_t)$ is an indicator function which is equal to 0 if the system is crowd-sensing, i.e., J_t= 1 else it is 1. Here, e_0 denotes the minimum energy needed by the smartphone to remain in active state below which the device would stop functioning. e_{cr} denotes the energy consumed by crowd-sensing application, e_1 and e_2 represent the energy consumed by the system when it is lightly or heavily loaded respectively. At any time slot t with system state X_t the probable next slot X_{t+1} is predicted with probability $P_{L_t \to L_{t+1}} \times P_{A_t \to A_{t+1}} \times P_{Y_t \to Y_{t+1}}$. P matrix records the state transition probabilities of each system from state X_t to next possible state X_{t+1} based on action performed and R matrix records the reward generated for this transition. Both matrices are represented by $m \times m$ dimension, where m is the number of system states.

Combining three state variables (L_t, A_t, Y_t) each system state as $X_t = (L_t, A_t, Y_t)$ is obtained where $P_{X_0 \to 2}$ denotes the state transition probability from 0 to 2 and $R_{X_0 \to 2}$ denotes the reward that is produced due to the change from state 0 to 2. Let $(L_{t_0}, A_{t_0}, Y_{t_0})$ and $(L_{t_2}, A_{t_2}, Y_{t_2})$ represent state 0 and state 2 respectively. The elements of P matrix, $P_{X_0 \to 2}$ is specified by the probabilities of state transition of each state variable from 0 to 2, i.e., $P_{X_0 \to 2} = P_{L_{t_0 \to 2}} \times P_{A_{t_0 \to 2}} \times P_{Y_{t_0 \to 2}}$ and $R_{X_0 \to 2}$ gives the reward which is resulting of action at performed at state 0 $(L_{t_0}, A_{t_0}, Y_{t_0})$ to state 2 $(L_{t_2}, A_{t_2}, Y_{t_2})$. Thus, a pair of P and R matrix is required for each action. This MDP formulation is solved using value iteration technique [112]. For any stationary policy $\pi = (\pi_0, \pi_1, ..)$, the state value function at a state $x \in X$ satisfies the Bellman equation [112],

$$V^{\pi}(x_t) = R(x_t, \pi(x_t)) + \gamma \sum_{x_{t+1}} P(x_{t+1}|x_t, \pi(x_t)).V^{\pi}(x_{t+1}) \tag{4.10}$$

Value iteration function takes into account P, R, and discount factor γ as arguments and assigns an arbitrary value V_0 to each state which is repeated for all state x. In next iteration $V_{z_{t+1}}$ is computed by Bellman backup at x and the iterations are continued until $max_x|V_{z_{t+1}} - V_z| < \epsilon$ (i.e. ϵ convergence). This value iteration technique results in number of iterations and discounted utility values $U[(r_0, r_1, r_2...)] = r_0 + \gamma.r_1 + \gamma^2 r_2 + ...$

The workflow of the approach and its implementation is summarized in flowchart Fig. 4.2. Here initial probability values of the device in a particular state are assumed. After formulation of MDP, probability matrix P and reward matrix R are fed to the value iteration technique along with ϵ and the discounting factor γ to obtain three values namely, maximum utilization for a state, the decision and the time taken to reach the goal. For better understanding of the above process let us take an example where p_{on}= 0.5, p_{off} = 0.6, q_{on} = 0.7, q_{off} = 0.8 and $|L|$=4 (thus, l_t = {0,1,2,3}). These values are taken in order to show better variation in output

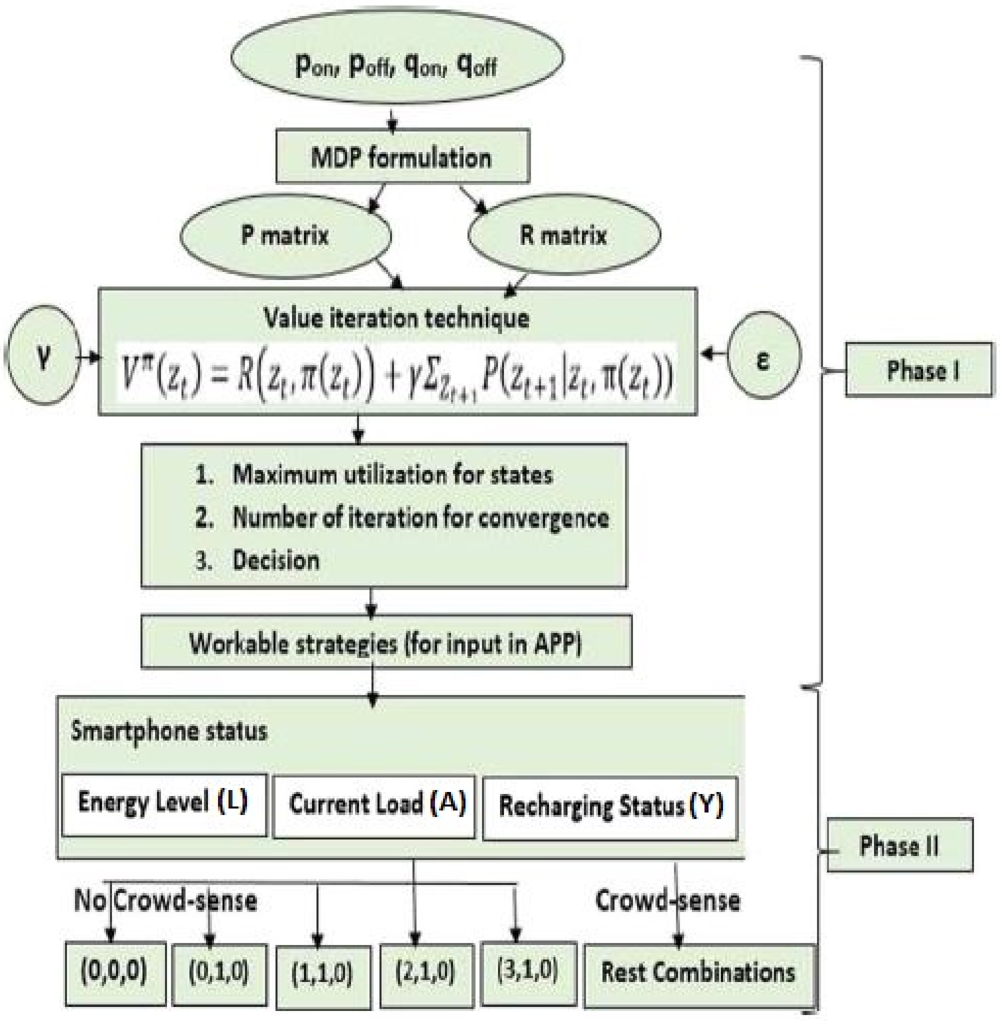

$p_{on}, p_{off}, q_{on}, q_{off}$
MDP formulation
P matrix
R matrix
Value iteration technique
$V^{\pi}(z_t) = R(z_t, \pi(z_t)) + \gamma \Sigma_{z_{t+1}} P(z_{t+1}|z_t, \pi(z_t))$
γ
ε
Phase I
1. Maximum utilization for states
2. Number of iteration for convergence
3. Decision
Workable strategies (for input in APP)
Smartphone status
Energy Level (L)
Current Load (A)
Recharging Status (Y)
Phase II
No Crowd-sense
Crowd-sense
(0,0,0)
(0,1,0)
(1,1,0)
(2,1,0)
(3,1,0)
Rest Combinations

Figure 4.2: Flowchart of the crowd-sensing process and its implementation in smartphones

depending on input conditions. Any other values in the range (0.5, 1) would be fine. Given the input parameters, the probability of a smartphone to be heavily loaded or lightly loaded in the next time slot is as follows according to Eq. 4.7. Thus, strategic decisions could be made based on this precalculated policy in order to get long term benefit in terms of crowd-sensing incentives and energy consumption. Workable strategies are formulated from the decision obtained from MDP simulated process. The smartphone application may tune to the optimal decision to crowd-sense according to the working conditions. The output of this strategy can be fed to any smart device which needs to strategically decide when to crowd-sense for

Table 4.2: Reward for different states for p_{on} = 0.5, p_{off} = 0.8 , q_{on}= 0.7 , q_{off} = 0.5

State ($L_tA_tY_t$)	Reward (J_t =1)
000, 100, 200, 300, 010, 110, 210, 310	0
001, 101, 201, 301	2
011, 11, 211, 311	1.286

opportunistic crowd-sensing applications.

The reward is awarded to a node depending on if the node has participated in crowd sensing. Table 4.2 summarizes the reward for a smartphone when it is crowd-sensing, calculated following Eq. 4.3 where the values of the input parameters are $p_{on} = 0.5$, $p_{off} = 0.8$, q_{on}= 0.7 , $q_{off} = 0.5$. This shows how the MDP is formulated and solved following the steps summarized in Fig. 4.2.

4.3 Designing crowd-sensed application for indoor localization

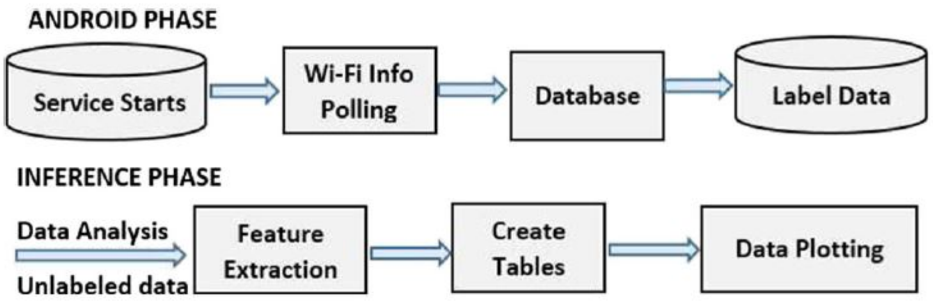


Figure 4.3: Mobile crowd-sensing framework for indoor localization system

To implement and check the effectiveness of the strategy, a simple but relevant crowd-sensing application is considered. A brief description of the said application is given here. Indoor localization revolves around the idea of locating an individual through the RSSI data received from wireless APs around that area. The server, upon getting this data, predicts the location of the individual based on previous fingerprint RSSI data [115]. The MCS application collects Wi-Fi information from the smartphones of the crowd at a given area and sends that data to a database server. This data, known as fingerprint data, greatly reduces the burden to repeatedly collect fingerprint upon any changes in the experimental area. Updated fingerprint dataset at the server side is used to build the training model. In the inference phase, by collecting Wi-Fi information from a device, the test data set is formed and with the help of the training model, the location of the device can be inferred. This is shown in Fig. 4.3. The first phase is focused that uses MCS framework to collect fingerprint data from the crowd and design a data collection strategy for the devices of the crowd so that they can optimally lend their devices to crowd-sensing depending on their device usage and other conditions. The MCS application named APSense is developed for Android based smartphones. Initially, the user is given the following options to choose from.

- scan: for obtaining a fresh set of data from the environment;
- view data folder: for obtaining data that has been previously crowd-sensed and stored in a structured file.
- receive: receiving the csv data file through email.

In order to check the effectiveness of this MDP based strategy summarized in Fig. 4.2, the proposed strategy is compared with three intuitive strategies, continuous sensing, random sensing and sensing according to Poisson distribution.

4.4 Implementation of various crowd-sensing strategies through indoor localization

Indoor localization based on Wi-Fi (IEEE 802.11 WLAN standard) signal strength or RSSI has become a prominent approach [115] nowadays. Here the smartphones of citizens scan for Wi-Fi signals from available access points in the experimental region and send it to a server for further processing. These collected data are then preprocessed to form training sets that can later be used to predict the location of a smartphone user in that area. The main problem toward scalable indoor localization is the huge effort needed to fingerprint every location for different environmental conditions and thus indoor localization is selected as a case study of MCS application in this work. Here, the Android application can crowd-sense for indoor localization application. It collects Wi-Fi fingerprints and sends it to a server. Performance of the MDP based strategies incorporated in the application is analyzed and also compared with other intuitive strategies including stateof- the-art strategies in [63] for similar experimental setup.

The entire process of implementing crowd-sensing strategy following MDP is carried out in two phases; first in Phase I i.e. before deployment probable states are defined (as in Eq. 4.1) and accordingly, state transition probabilities are evaluated. In this work, four energy levels i.e. $L_t \in \{0, 1, 2, 3\}$, $A_t \in \{0, 1\}$ and $Y_t \in \{0, 1\}$ are considered corresponding to each action $J_t \in \{0, 1\}$. To construct the probability matrix (P) for each action A_t, total $(4 \times 2 \times 2) = 16$ probable system states are considered. Subsequently, the reward matrix (R) is constructed by estimating rewards following Eq. 4.3 based on action A_t performed at each state X_t. Next, the value iteration process is performed using R [2] simulator which is a free software environment for statistical computing and graphics. P and R along with discount factor γ are given as input to the value iteration function to obtain the series of actions J_t as output for which cumulative reward value measured in terms of discounted utility gets maximized. The output acquired in Phase I, i.e., the decision obtained is then used to find a workable strategy and that is passed on as input in Phase II, i.e., smartphones, and setting parameter values in APSense for MDP based strategy.

[2]https://www.r-project. org/

4.5 Experimental setup

In this work, five different smartphone configurations are used for experimentation: Moto G4 plus with the battery capacity of 3050 mAh having 3GB RAM and using version 7.1.2 version of Android, Redmi note 4, 4100 mAh battery capacity and 4GB RAM using Android 7.0 version, Samsung Galaxy Tab E with 5000 mAh battery capacity and 1500MB RAM using Android 4.4.4 version, Yuphoria YU5010A with battery capacity of 2230 mAh having 2GB RAM and using version 5.1.1 version of Android and Samsung Grand Prime having Android version 5.0.2, 2500mAh battery capacity and 1GB RAM. The application is executed 10-20 times to obtain the amount of energy consumed.

For these experiments, L_t of Eq. 4.1 has been mapped to different energy states 0, 1, 2, and 3 corresponding to battery percentage above 20%, 40%, 60% and 80% respectively. As energy consumption increases greatly when CPU usage exceeds 50% for many smartphone configurations, so, A_t in Eq. 4.1 is 1 when CPU usage is more than 50% and is 0 otherwise. Y_t is 1 if the device is recharging and is 0 otherwise. In real-life scenario, crowd-sensing attempts depend on various factors like battery status, network availability, computational load etc. In this context, different distributions of sensing actions, that is, strategies are explored that may likely occur in practice. For all the strategies present in the application, the resultant data is stored in a csv file. The application also records the frequency of the crowdsensing attempts. As mentioned earlier four strategies are used:

- Continuous crowd-sensing strategy: In this strategy, APSense continually collects Wi-Fi information for the entire duration of the experiment disregarding the usage behaviour of the device.

- Random strategy for crowd-sensing: In this mode of the operation, the application lends itself to crowd-sense and send collected Wi-Fi information, but the frequency of the scan is decided following a Uniform distribution.

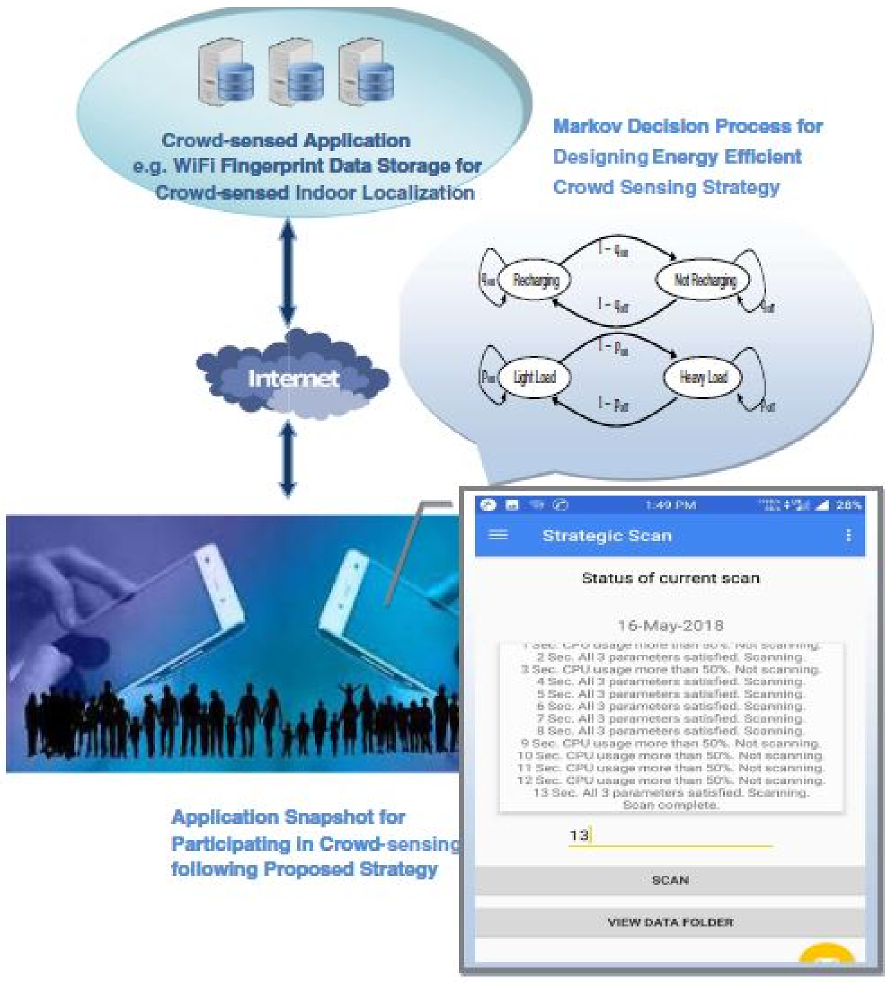
- Crowd-sensing following Poisson distribution: The application lends itself to crowd-sense and send collected Wi-Fi information to the server according to a Poisson process. At every second, the Poisson process decides if crowd-sensing can be invoked.

- Strategy Based on MDP: A device invokes crowdsensing according to the flowchart Fig. 4.2. Thus, APSense collects WiFi fingerprints following the strategies obtained by solving MDP. For instance, crowd-sensing action yields good reward if the battery percentage of the device is above 20% and the CPU usage is less than 50%. Such conditions, along with other strategic conditions are checked every second and if and only if the conditions are satisfied crowd-sensing is invoked.

Fig. 4.4 shows a screenshot of the application where data are collected according to MDP based strategy for 13 seconds. Whenever the CPU usage is more than 50%, the device does not lend itself to crowd-sense according to the strategy obtained from Phase I as shown in Fig. 4.2. As soon as the crowd-sensing procedure starts the application gives a simple feedback. A notification is shown when a crowd-sensing operation is being requested to perform while another crowd-sensing operation is already running in the background. In the case where the Wi-Fi is not turned on the application gives one toast to notify the user to turn it on. The RSSI values of the wireless APs in the range are stored in a csv file and later when connected to the Internet, the value is uploaded to the server. The application runs in the background and all other activities can be performed normally when this application runs.

The library used for transmitting data to the server is Volley [3]. Volley is an HTTP library that makes networking for Android applications easier and most importantly, faster. Two PHP scripts are designed for the connection. One for making the connection between Android application and server, another one for inserting

[3]https://developer.android.com/training/volley/



```
1 Sec. CPU usage more than 50%. Not scanning.
2 Sec. All 3 parameters satisfied. Scanning.
3 Sec. CPU usage more than 50%. Not scanning.
4 Sec. All 3 parameters satisfied. Scanning.
5 Sec. All 3 parameters satisfied. Scanning.
6 Sec. All 3 parameters satisfied. Scanning.
7 Sec. All 3 parameters satisfied. Scanning.
8 Sec. All 3 parameters satisfied. Scanning.
9 Sec. CPU usage more than 50%. Not scanning.
10 Sec. CPU usage more than 50%. Not scanning.
11 Sec. CPU usage more than 50%. Not scanning.
12 Sec. CPU usage more than 50%. Not scanning.
13 Sec. All 3 parameters satisfied. Scanning.
Scan complete.
```

Figure 4.4: Snapshot of the application while participating in crowd-sensing according to the proposed strategy

data to database table at the server end. The PHP script for data insertion is stored in a String variable inside Constants class.

4.6 Results and discussions

Here, a set of experiments are carried out first to study the performance of Phase I followed by that of Phase II.

4.6.1 Results of Phase I

In the first Phase I, the simulation is carried out in R [4] which is a software for statistical computing. MDP formulation is fed as input to value iteration. From

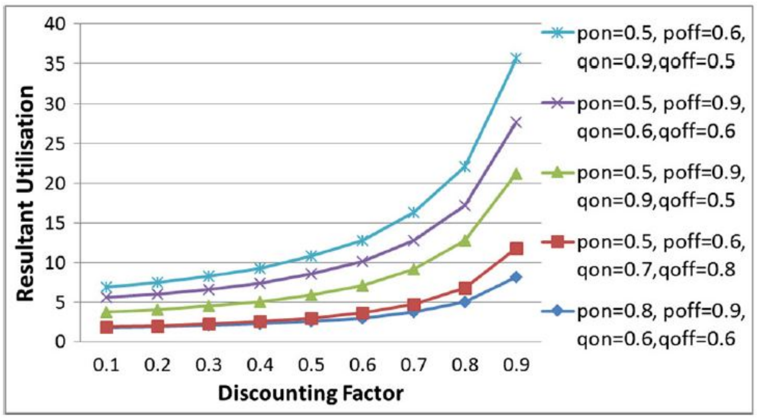


Figure 4.5: Resultant utilization corresponding to varying discounting factor

Fig. 4.5 it can be observed that while varying the values of p_{on}, p_{off}, q_{on} and q_{off}, for each and every case, the resultant utilization is maximized when discounting factor γ is 0.9. Discounting factor also plays a crucial role as depicted in Fig. 4.6. With higher discounting factor, a node is expected to take a decision much aligned with its goal and hence, all states are found to yield better resultant utilization. For clarity, only a few representative states are shown in the figure though all other states are found to follow similar pattern. So, for the experiments stated above, the discounting factor is kept at 0.9. In the value iteration approach, the number of iterations required to converge is polynomial [112] in the number of states and the magnitude of the largest reward, thus the computation complexity of each iteration is $O(|X|^2|J|)$, where the discount factor is held constant. However, in the worst case the number of iterations grows polynomially in $\frac{1}{1-\gamma}$, so the convergence rate slows considerably as the discount factor approaches 0.9.

Rewards obtained in each step due to performing actions contribute to the utility calculation in a way that, sooner rewards have higher utility than later rewards.

[4]https://www.r-project.org/

Table 4.3: Example of Strategic Decision of a node for given input parameter values

State($L_tA_tJ_t$)	Decision
000	No Crowd-sense due to insufficient remaining energy
001, 101, 201, 301	Crowd-sense as system recharging.
010, 110, 210, 310	No Crowd-sense due to current load on the system
011, 111, 211, 311	Crowd-sense as system recharging
100, 200, 300	Crowd-sense as there is no load and high remaining energy

Thus with higher values of the discounting factor, a particular node is able to take a decision which could be beneficial in the long run. So, for the subsequent experiments, γ is kept at 0.9. Again q_{on} is varied to find out the resultant utilization

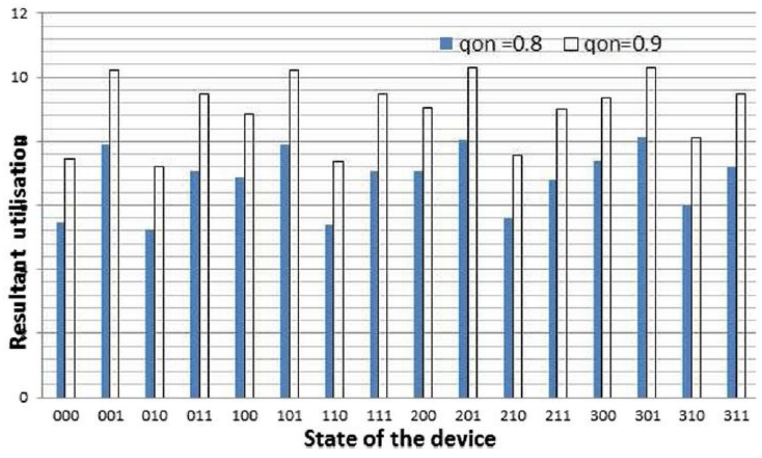


Figure 4.6: Resultant utilization corresponding to each state of MDP formulation while varying q_{on}(probability of recharging): A state X_t is represented as (L_t, A_t, Y_t) along X-axis

for each of the states. From Fig. 4.5 it can be observed that when the device is recharging ($Y_t = 1$) resultant utilization is better, irrespective of the current load or the remaining energy of the device. However, when not recharging, current computational load, which is CPU utilization is found to play a key role. In the proposed strategy it can be found that in spite of the device not recharging, the device having load still perform well if the probability of recharging is high. Even under low load conditions, a device may still crowd sense if it has adequate battery power.

Table 4.3 summarizes the decision taken by a node at a given state for p_{on} =

0.5, $p_{off} = 0.8$, $q_{on} = 0.7$, $p_{off} = 0.5$, where the justification behind each decision is also discussed. Extensive simulation has also been performed by taking different values of these parameters. It is found that by varying these parameter combinations similar type of results were still obtained as output. From extensive experiments varying the input parameters, it is found that, the most important factor for a node to be ready to participate in crowd-sensing is the battery recharge (Y_t). When the node is being recharged externally irrespective of whether there is load in the node at that particular time instant or the battery power is low, the node is bound to crowd-sense (since the reward is good enough). The output acquired in Phase I is passed on as input to Phase II while developing the Android application.

4.6.2 Results of Phase II

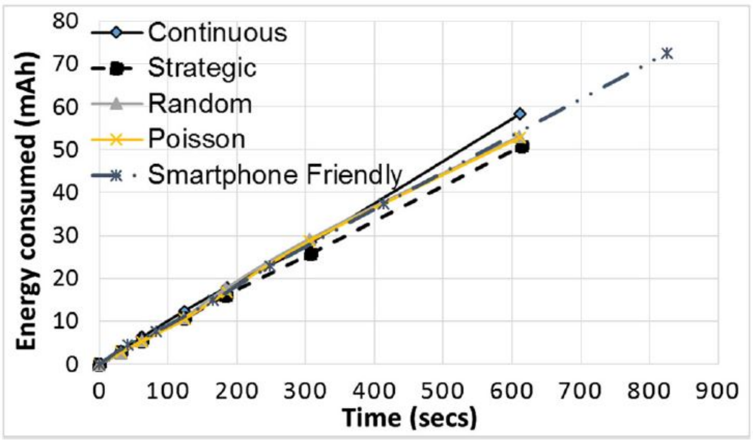


Figure 4.7: Energy(mAh) consumed by a mobile device per unit time(sec) for five different strategies using the device Yuphoria YU5010A

The performance of the Android application is analyzed for three intuitive non-MDP policies (Continuous crowdsensing, Random crowd-sensing, Crowd-sensing following Poisson Distribution) with MDP based strategy and Smartphone friendly policy as in [63]. In most of the experiments, better results are obtained in case of MDP based strategy. In Fig. 4.7 energy consumed by a mobile device is plotted when it lends itself to crowd-sense following a particular strategy with respect to time. The Energy Consumed (EC) can be broken into two factors namely:

- The number of scans performed (n), i.e., the sensing energy (during time t)
- Computation power consumed (cp) to compute the strategy (during time t)

$$EC = n \times (energy\ \ consumed\ \ per\ \ scan) + cp \times t \tag{4.11}$$

Here t is the time in seconds. The value of energy consumed is directly proportional to the number of times crowd-sensing is performed plus some other factors which can be treated as constant. The energy consumed by continuous crowd-sensing approach (similar to Collector friendly policy discussed in [63] is more than other approaches as it continuously crowd-senses at every time instance irrespective of the conditions of the device. Thus the first term in Eq. 4.11 always supersedes. As time increases, energy consumed by strategic crowd-sensing is found to be the

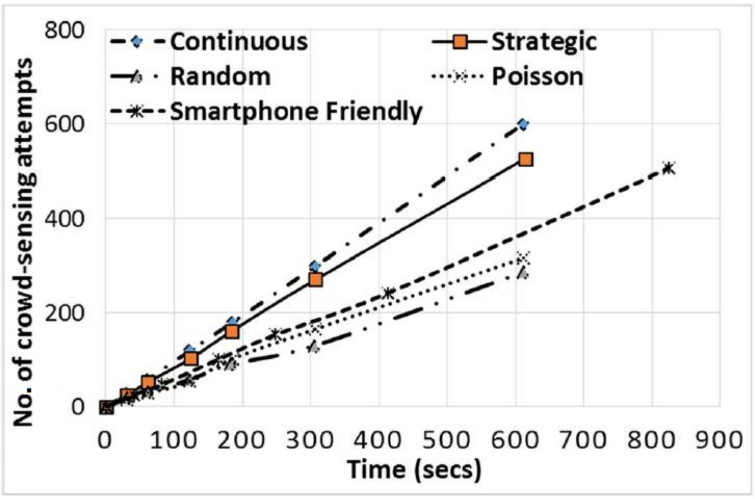


Figure 4.8: Number of attempts made by the device Yuphoria YU5010A to crowd-sense per unit time (secs) for various crowd-sensing strategies

least with respect to other four strategies as MDP takes into account the present condition of the device while deciding to crowd-sense. The advantage of considering device conditions while deciding to crowd-sense is aptly indicated when the number of corresponding crowd-sensing attempts are plotted.

The graph in Fig. 4.8 shows the number of times crowdsensing is performed with respect to time. The number of times a device crowd-senses for continuous strategy is maximum owing to the fact that the device ceaselessly crowd-senses without taking into considerations the various device parameters. Following Poisson and Random strategies, a device crowd-senses according to the respective probability distribution. The attempts to crowd-sense made by Smartphone friendly policy [63] is almost continuous under favorable conditions up to a certain limit defined in [63]. Then it stops crowd-sensing thus resulting in less number of scanning attempts. From Figs. 4.7 and 4.8 it can be observed that, for the above experiments, MDP based strategic crowd-sensing results in better trade-offs between energy consumed and number of crowd-sensing attempts than any other approaches.

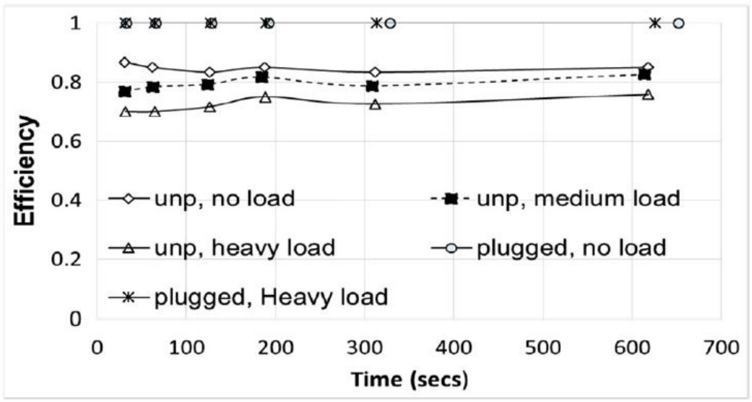


Figure 4.9: Crowd-sensing efficiency for various CPU load conditions with respect to time using Samsung Grand Prime

The role of device conditions on strategic crowd-sensing is investigated in the next experiment. Results are plotted in Fig. 4.9 for varying computational load conditions as detailed in Table 4.4. The table shows that the exact value of CPU utilization for a given load condition depends on device configurations though the ranges are not overlapping. The crowd-sensing efficiency is defined as follows:

$$efficiency = \frac{crowd - sensing\ invocation}{crowd - sensing\ attempt} \tag{4.12}$$

The application periodically attempts to crowd-sense for the entire duration of the experiment. If the device conditions are found suitable according to the strategy

Table 4.4: Average CPU utilization by Android devices for different computational load conditions

Load Conditions	Applications Running	Average CPU load (%)				
		Motog4 plus	Redmi Note5	Samsung Grand Prime	Samsung Galaxy Tab E	Yuphoria
unplugged, no load	idle	15.12	12.77	10.37	10.69	10.51
unplugged, medium load	pdf & image viewers, social media	25.44	24.25	30.99	29.14	23.80
unplugged, heavy load	multimedia streaming	43.46	39.62	46.64	45.32	37.89
plugged, no load	idle	12.22	11.79	7.98	8.59	6.69
plugged, heavy load	multimedia streaming	31.25	29.12	36.98	35.12	26.96

described in Fig. 4.2, then only APSense actually lends itself to crowdsensing. So, the efficiency of 100% can only be obtained when the device is recharging according to the policy reflected in Fig. 4.5. A mobile device is found to lend itself more

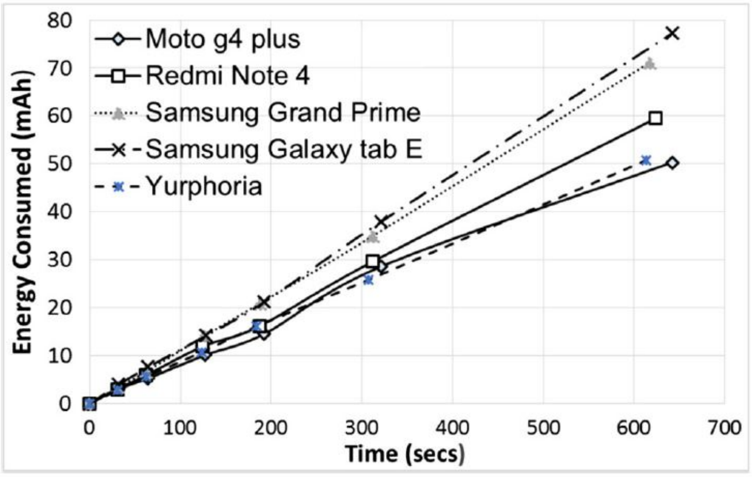


Figure 4.10: Energy(mAh) consumed by various smart devices with time (secs) for load condition-(unplugged, no load)

frequently to crowd-sensing when the load on the device is less and reduces the crowd-sensing activity as the load in the device starts increasing. This mechanism of reducing the activity of crowd-sensing with increased load in the device helps in the power management and keeping the device active for longer period of time. Fig. 4.10 shows the corresponding energy consumption when the devices are unplugged and idle. The devices having more number of sensors result in more energy consumption due to crowd-sensing. Thus with time, power consumption scales up for all the devices as more crowd-sensing attempts are made but managing the power becomes utmost for devices rich in sensors.

4.7 Summary

Mobile crowd-sensing applications often consume an appreciable amount of devices' energy. So in this work, MDP based formulation for energy efficient crowd-sensing strategy is presented. The proposed approach is implemented in an Android application that collects crowd-sensed data for indoor localization. Performance of proposed algorithm is compared with other intuitive strategies and state-of-the-art strategy as in [63] using different devices. The MDP based strategy is found to yield efficient crowd-sensing for different load conditions. Not only data but tasks can also be offloaded. However, when the devices need to offload the data/ tasks to the cloud, various types of faults are encountered. These are discussed in the next chapter.

List of Publications:

Conference:

1. **Ray, A.**, Chowdhury, C. and Roy, S., 2017, December. Strategic decision for crowd-sensing: An approach based on Markov decision process. In 2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS) (pp. 1-6). IEEE.

Journal:

1. **Ray, A.**, Chowdhury, C., Mallick, S., Mondal, S., Paul, S. and Roy, S., 2020. Designing energy efficient strategies using markov decision process for crowd-sensing applications. Mobile Networks and Applications, 25, pp.932-942.

Chapter 5

Fault-tolerant Approach to Reduce Failures in Offloading Systems

In case of mobile crowd-sensing and computing, data processing and/or task processing takes place at the edge or the cloud servers. Transferring of tasks by the mobile devices (or offloadee) to the resource rich computational devices such as edges or clouds, called surrogate, for processing of these data is known as cyber foraging [68] or offloading [69], where the backend servers (surrogates), plays an important role by providing the computation platform along with the resources needed for computation of the tasks. Thus, this chapter discusses the issues related to fault and failures in offloading systems. The objectives and contributions as follows:

Problem description Identifying different types of faults and classifying the faults that are encountered while data is being offloaded to the backend servers. Also proposing of different techniques to tolerate the faults and provide smooth and seamless services during the process of offloading.

Objective In order to handle the faults, checkpoints are necessary, but if the checkpoints are retained in the mobile device then the performance of the devices may be severely affected. A cost effective technique is the need of the hour to taper down the usually occurring faults that obstructs the functioning of the offloading

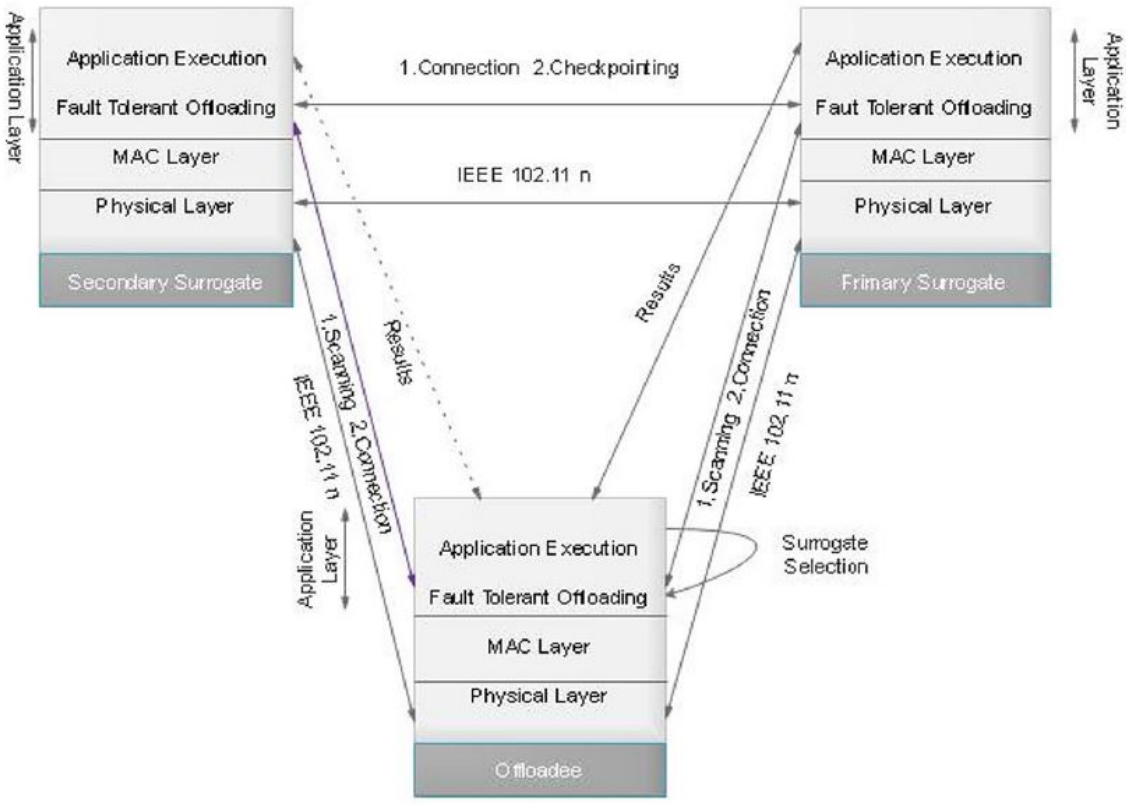


Figure 5.1: Architecture of offloading system

systems. The objective of this chapter is to discover and classify the sets of faults that may be encountered in the process of offloading and proposing techniques to tolerate the faults.

Contribution The contribution of this chapter is proposing an efficient framework which helps in detecting a fault, if occurs, classifying the faults in the offloading systems and proposing ways to prevent certain types of failures. The framework also proposes technique to tolerate the faults to a certain degree.

The rest of this chapter is organized as follows: Section 5.1 discusses the proposed approach, where Section 5.2 discusses the implementation of the proposed approach. In Section 5.3 experimental setup is stated and Section 5.4 discusses the results and Section 5.5 summarises the whole chapter.

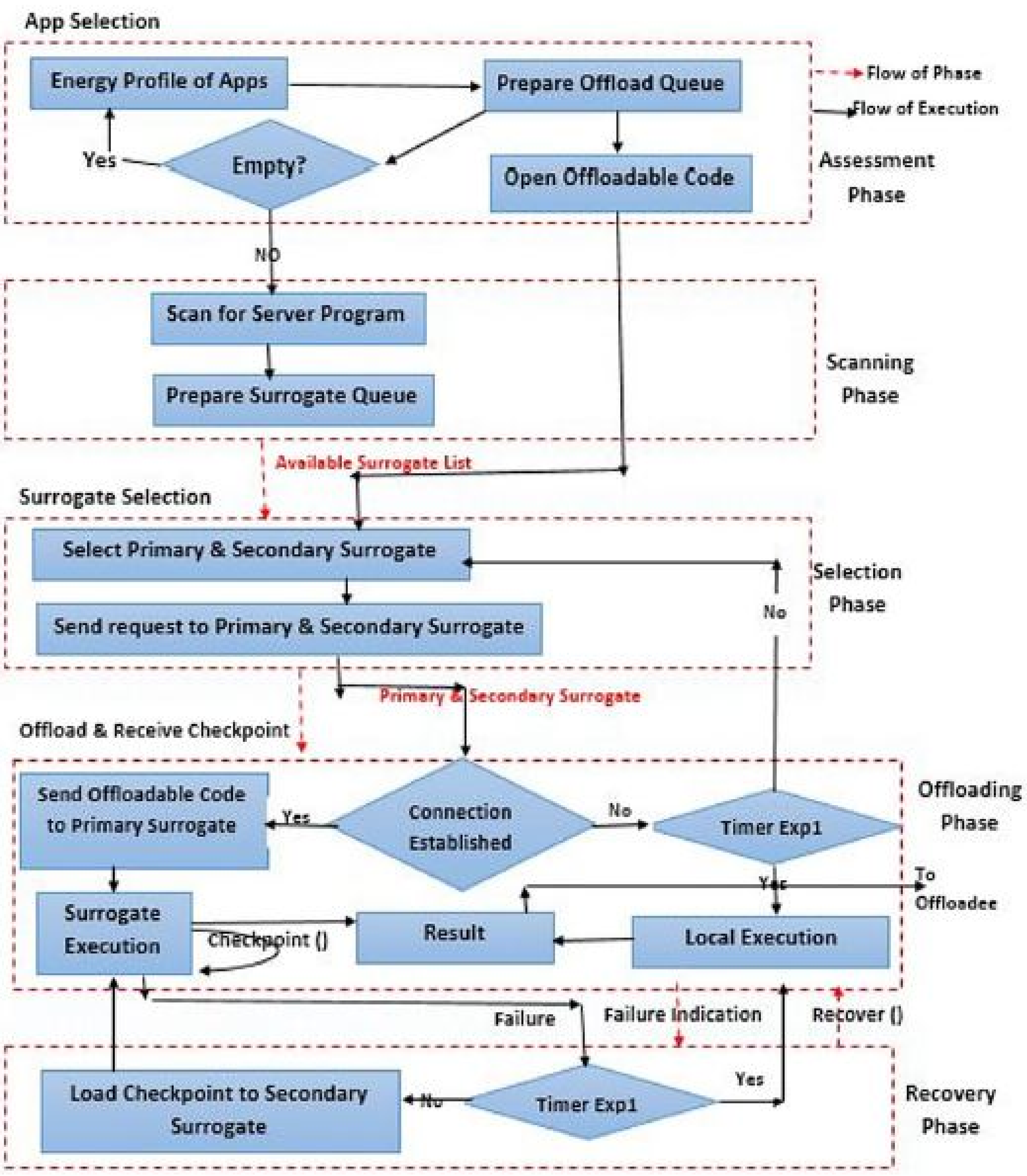


Figure 5.2: Workflow of the proposed offloading framework

Table 5.1: Commonly used notations

Term	**Significance**
$T^{exec}_{S_i}$	Expected execution time at surrogate S_i
T_{S_i}	Expected overall response time for surrogate S_i
T_w	Maximum waiting time for getting response from any primary secondary surrogate pair
$T_{timeout}$	Time out period for secondary surrogate
$T^{comm}_{S_i,off}$	Expected communication time between S_i and offloadee
$T^{comm}_{S_p,S_s}$	Expected communication time between primary and secondary surrogate
T_{max}	Total time-out interval for offloading
Surrogate Parameters	
S	Set of available surrogate
S_p	Primary surrogate
S_k	Any surrogate k not primary
S_{Sel}	Selection of j suitable surrogates by offloadee
S^{rs}_k	No of successful offloading response by surrogate S_k
S^{rf}_k	No of times surrogate k failed
S^{trust}_k	Trust of surrogate k
S^{SR}_k	Success rate of surrogate k
S^{A}_k	Availability of surrogate S_k
Offloadee parameters	
req^k_n	Number of offloading requests sent to S_k
res^k_n	Number of successful offloading results from S_k

5.1 Proposed approach: Fault-tolerant offloading framework

The proposed fault-tolerant offloading framework is based on the architecture in Fig. 5.1. The region of implementation can be an institute campus or an office which utilizes Wi-fi for communication between the surrogate and the offloadee that is supporting IEEE 802.11 WLAN. Usually resource rich servers which are idle in the office or institute campuses are selected as surrogates which has Windows or Linux operating system, on the other hand, the smart handheld devices having Android operating system is considered to be offloadee. Thus, an edge server residing closer to the Android devices could act as a surrogate. Depending on the condition of the network, the availability of the surrogates, and the application, the fault-tolerant offloading layer of the system takes the decision whether to offload the tasks to the surrogates or to execute the tasks at the device itself. In general, it is preferred that the I/O based applications be executed in the devices and computation intensive, time-sensitive applications be offloaded to the surrogates. When a task is to be offloaded then the offloading layer looks for the available surrogates (at the edge of the network) that is connected through Wi-Fi. Fault-tolerant offloading layer, in this architecture, seeks only those surrogates in the network which has the potential to handle and tolerate failure situations when one arises. Terms used in this chapter are summarized in Table 5.1. The surrogates are evaluated based on its performance and allotted a parametric value which they advertise periodically. It is considered that there are N number of offloadees and K number of surrogates. Therefore, n denotes an offloadee and k denotes a surrogate. The different phases of the framework are: 1) Assessment phase, 2) Scanning Phase 3) Selection phase, 4) Offloading Phase and optional 5) Recovery Phase as described in Fig. 5.2.

1. Assessment Phase: In this phase, the tasks are evaluated based on the task requirements. If the requirements of the tasks are more of interactions and I/O involvement then message exchanges are more. In such scenario, sending such tasks for offloading will lead to more overhead leading to more energy consumption. On the other hand, in case of tasks being computation intensive, and the energy consumed for executing the tasks at device is more than the energy consumed for offloading the tasks and receiving the results, it is better to offload those tasks to the surrogates. Hence, tasks assessment phase is an important phase in this framework. If there is any tasks in the offloading queue, then the next phase, i.e., scanning phase gets activated.

2. Scanning Phase: Nearby surrogates are scanned for offloading the tasks. Depending on the parametric values the surrogates have advertised, they are put in the surrogate queue seen from Fig. 5.2. Each surrogate k keeps a record of parameters (S_k^{rs}, S_k^{rf}). The availability of the surrogates is calculated by:

$$S_k^A = \frac{S_k^{rs}}{S_k^{rs} + S_k^{rf}} \tag{5.1}$$

where S_k^A nearing one indicates the high availability, and this value is broadcasted during offloading sessions.

3. Selection Phase: Depending on the value of S_k^A of the surrogates, and previous interaction with the surrogates, in the known neighbourhood, the offloadee chooses a set of surrogates S_{sel}. When a task is to be offloaded, requests are sent to the surrogates from the selected set of surrogates S_{sel}. The framework selects a primary surrogate where the tasks is to be offloaded and executed and a secondary surrogate which is kept for backup when needed.

4. Offloading Phase: The offloadee chooses two time out values during the session of offloading: probable execution time ($T_{S_i}^{exec}$) where $S_i \in S_{sel}$ and probable communication time ($T_{S_i,off}^{comm}$) for transmission of data and results between the offloadee and surrogate S_i. The total probable response time when a job is offloaded is as follows.

$$T_{S_i} = T_{S_i}^{exec} + T_{S_i,off}^{comm} \tag{5.2}$$

When the offloadee has tasks which are computation intensive, the offloadee takes the decision to send off the tasks to the surrogates. Between the available surrogates, the offloadee selects one primary and one secondary surrogate. The offloadee awaits for T_{S_i} time for results after tasks completion, from the surrogate. The maximum waiting time T_w that the offloadee waits is as follows:

$$T_w = max\{T_{S_i}\} \forall S_i \in S_{sel} \tag{5.3}$$

5. Recovering Phase: The offloadee selects another set of primary secondary surrogate pair, and send off the task, if the result is not received by the offloadee within the time T_w and $T_w < T_{max}$ and awaits till T_w time. Until time T_{max} has passed by, the offloadee may attempt a number of times to get the tasks

done by the surrogates, where

$$T_{max} = \sum_{S_i \in S_{sel}} T_w \tag{5.4}$$

The offloadee executes the task locally if the result does not arrive within the time T_{max}.

The notion of timeout and redundancy is implemented here. The offloadee evaluates the Success Rate (S_k^{SR}) for each surrogate k and request n, depending on the "Offloadee Parameters" from Table 5.1 as follows:

$$S_k^{SR} = \frac{res_n^k}{req_n^k} \tag{5.5}$$

The S_k^{trust} of every surrogate the offloadee has dealt with is evaluated by the offloadee, at a particular location by the following:

$$S_k^{trust} = \begin{cases} \alpha \times S_k^{SR} + \beta \times (1 - (S_k^{ResTime}/T_i)) & \text{if } \; S_k^{ResTime} < T_i \\ \alpha \times S_k^{SR} & \text{otherwise} \end{cases} \tag{5.6}$$

where $\alpha + \beta = 1$. The offloadee keeps a record of all the surrogates(k^s) trust values that it has communicated for a particular location. From the list of surrogates S_{sel}, the highest trust value and most available surrogate is assigned the offloaded task. Hence, for selection of the most appropriate surrogate, historical data of offloadee and surrogate is needed.

The algorithm running at the offloadee is summarized as Algorithm 1. The offloadee broadcasts the necessity of surrogates so that the task can be offloaded for execution, when there is any offloadable task. If the requirements list of the offloadee is satisfied by any surrogates, they reply back with their availability as response. The offloadee creates a list of surrogates S_{sel} and chooses a primary surrogate(s) $S'_p \in S_{sel}$

Algorithm 1 Offloadee_Code

```
 1: input: Primary surrogate $S'^{p}$;
 2: output: result of the job assigned by offloadee to surrogate/ result of
    the job done by the offloadee;
 3: broadcast offloading ($S$)
 4: waitforEvent();
 5: if ($Event(receiveAvailability)$) then
 6:     $S_{set}$ ← build Surrogate list()
 7: end if
 8: $S'^{p}$ ← Select Primary Surrogate ($S_{sel}, k$)
 9: $S_s$ ← Select Secondary Surrogates ($S_{sel}, S'_p$)
10: for $\forall s_p \in S'_p$ do
11:     send offloadingRequest ($s_p$)
12: end for
13: send offloadingNotification ($S_s$)
14: $startTimer()$;
15: $waitforEvent()$;
16: if ($Event(receiveResult)$) then
17:     if $Result\ arrives\ from\ multiple\ surrogates$ then
18:         selectResult();
19:     end if
20: end if
21: if ($Event(timeout)$) then
22:     locally execute the task
23: end if
```

Algorithm 2 Surrogate Code

```
1: input: request for Availability of the surrogate/ request for job of offloading done by Primary Surrogate/ request for Notification of the job of offloading done by Secondary Surrogate from checkpoint
2: output: status of Availability of surrogate resources/ result i.e., the completed job done by surrogate
3: waitforEvent();
4: if ($Event(receiveBroadcastOffloading)$) then
5:     $status \leftarrow$ check resources;
6:     if ($status == True$)) then
7:         sendAvailability();
8:     end if
9: end if
10: if $Event(receiveOffloadingRequest)$ then
11: $takeCkpt \leftarrow True$;
12: $startTimer()$;
13: $startExec()$;
14:     while ($takeCkpt$) do
15:         if $Event(CkptTimeout)$ then
16:             send Ckpt();
17:         end if
18:         if $Event(EndofTask())$ then
19:             notifyEndofTask();
20:             sendResult();
21:             $takeCkpt \leftarrow False$
22:         end if
23:     end while
24: end if
25: if $Event(receiveOffloadingNotification)$ then
26: $RevCkpt \leftarrow True$;
27: $startTimer()$;
28:     while ($RevCkpt$) do
29:         if $Event(RevCkpt)$ then
30:             store Ckpt();
31:         end if
32:         if $Event(notification\ (EndofTask())$ then
33:             stopTimer();
34:             $RevCkpt \leftarrow False$;
35:         end if
36:         if $Event(CkptTimeout())$ then
37:             startExec() from latest Ckpt;
38:             $sendResult()$;
39:             $RevCkpt \leftarrow False$;
40:         end if
41:     end while
42: end if
```

and a secondary surrogate(S_s) from the list. The offloadee waits for the result from the surrogates for T_{max} time, after sending the task execution request to both the surrogates, i.e., S_p and S_s. If no results are received, then the offloadee starts the task execution locally in its own device.

Algorithm 2 sums up the work done at the surrogates denoted by Surrogate_Code (S_s^{rs}, S_s^{rf}). The selected surrogate can be primary or secondary. If the selection is a primary surrogate, then the offloaded task is to be finished in a specific time. After a certain time interval, T_{Ckpt}, the primary surrogate $s_p \in S_p$ continues sending checkpoints to secondary surrogate, S_s, where in $ckpt(i, S_p)$, i is the index of checkpoint, and s_p is the primary surrogate where the checkpoint is noted. If in time interval T_{Ckpt}, the secondary surrogate S_s does not obtain any new checkpoint or response, the secondary surrogate S_s finishes the task from the last checkpoint, considering that the primary surrogate have crashed. The finished task is sent back to the offloadee.

Fig. 5.2. describes the summary of the workflow of the framework. If reliable delivery of results can be ensured from S_k or S_s then the offloadee crash can also be handled by this strategy.

At primary surrogate S_p the expected execution time taken by an offloaded task is denoted by $T_{S_p}^{exec}$. Here, it is assumed that the total time is split into small window of equal size T_{Ckpt} for checkpointing where $\sum T_{Ckpt} = T_{S_p}^{exec}$ and T_{S_p,S_s}^{comm} is the communication time between primary and secondary surrogate. Hence,

$$T_{timeout} = T_{Ckpt} + T_{S_p,S_s}^{comm} \tag{5.7}$$

Therefore, S_s times out after waiting for one checkpoint interval and the communication delay from S_p to S_s. To obtain the result, the offloadee waits for a maximum time of T_w, where $T_{timeout} < T_w$. Fig 5.5. describes the event. In, case of crash failure at primary surrogate, the secondary surrogate near vicinity completes

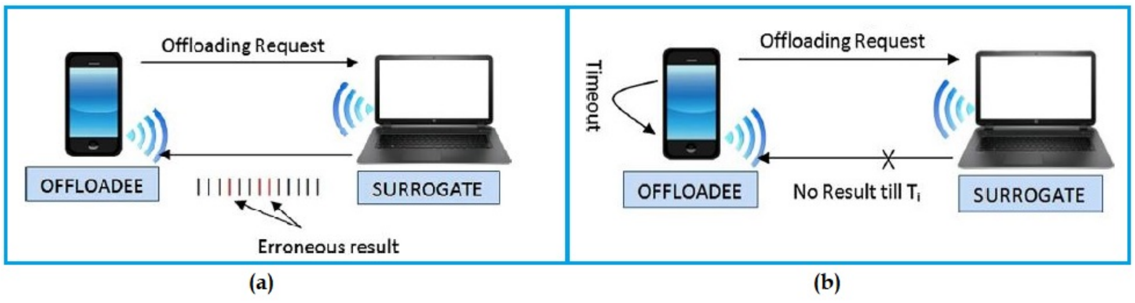


Figure 5.3: (a) Occurrence of omission and transient failure (b) Occurrence of crash failure

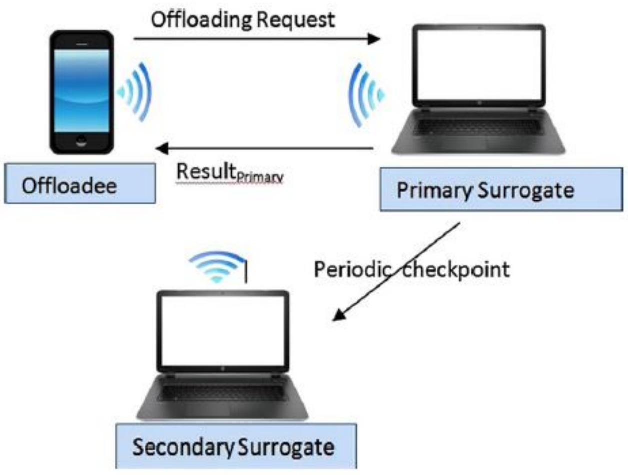


Figure 5.4: Offloading scheme depicting the responsibility of a primary surrogate, S_p and a secondary surrogate, S_s

the task and return the result back to offloadee, where the offloadee does not require to save the checkpoint. This is the beauty of this scheme.

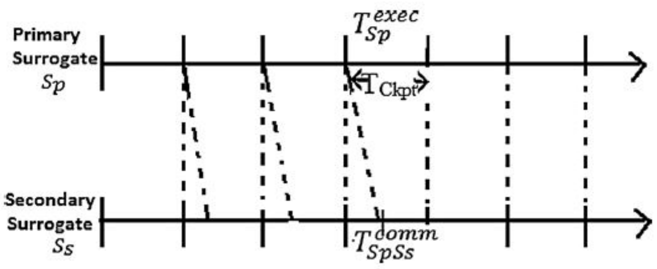


Figure 5.5: Timing diagram of occurrence of time out at primary surrogate

5.1.1 Detection of fault

On the basis of category of failure, faults can be detected which eventually leads to failure of the system. In this work, to detect faults which leads to crash, omission or transient failure, a combination of timeout and hashing mechanisms are being used.

- Omission failure: Surrogate unreachability or lossy network between the offloadee and the surrogates results in this type of failure in offloading systems. Omission failure can be identified by hash mechanism, when there is lossy network and some part of the result received by the offloadee may be erroneous. This type of failure can be detected by comparing the hash value calculated by the offloadee and the surrogate. In case of unreachability of the surrogate, the offloadee will not receive any result, and is similar to the crash failure condition.

- Transient failure: Similar to omission failure where the offloadee may receive some part of the result or erroneous result, this type of failure occurs when there is variations in environmental conditions. But in transient fault, on improvement of the link condition, the result sent by the surrogate is received correctly by the offloadee. Hash mechanism is utilized to detect this type of failure. Fig 5.3. describes this scenario.

- Crash failure: If there is no result obtained by the offloadee within time T_i from the surrogate, then crash failure has occurred as shown in Fig 5.3. and this can be identified by the timeout mechanism. The secondary surrogate S_s detects the crash of primary surrogate S_p by the time-out mechanism from Eq.5.7 and takes the responsibility to finish the task from the last checkpoint and sends it over to offloadee. For better handling of crash failure the following equation is performed:

$$\sum T_{Ckpt} < (T_i - T_{Ckpt} - T_{rec}) \tag{5.8}$$

One T_{Ckpt} interval is needed for detection of crash failure at primary surrogate by the secondary surrogate and T_{rec} time for recovery. The checkpointing overhead when expressed in time is $(T_{Ckpt} + T_{rec})$. It is assumed that,

$$T_{Ckpt} \approx T_{exec} \tag{5.9}$$

where, the total execution time at the surrogate without fault is denoted by T_{exec}. Those applications will be beneficial despite of faults, for which

$$T_{exec} + T_{Ckpt-overhead} < T_i \tag{5.10}$$

Fig 5.4. explains the handling of crash failure. Two surrogates are considered, one being primary another secondary. Single crash failure is tolerated as in case of either primary or the secondary surrogate, the other surrogate takes up the responsibility of task completion and sending back the finished task. In case, there is crash at offloadee, the surrogates stores the results and when the offloadee recovers from crash, the result is send over to the offloadee. However, when both the surrogates crashes, the offloadee detects by the $T_{timeout}$ time so it computes the task locally.

5.2 Implementation of fault-tolerant framework

A suitable application is selected for implementing this framework and computed in the following three modes:

1. Local execution: The task is computed locally in the offloadee and needs to be in Android device.

2. Offloading in absence of fault-tolerance: The task is executed at the surrogate when offloadee sends an offloadable task where the offloadee provides the “IP

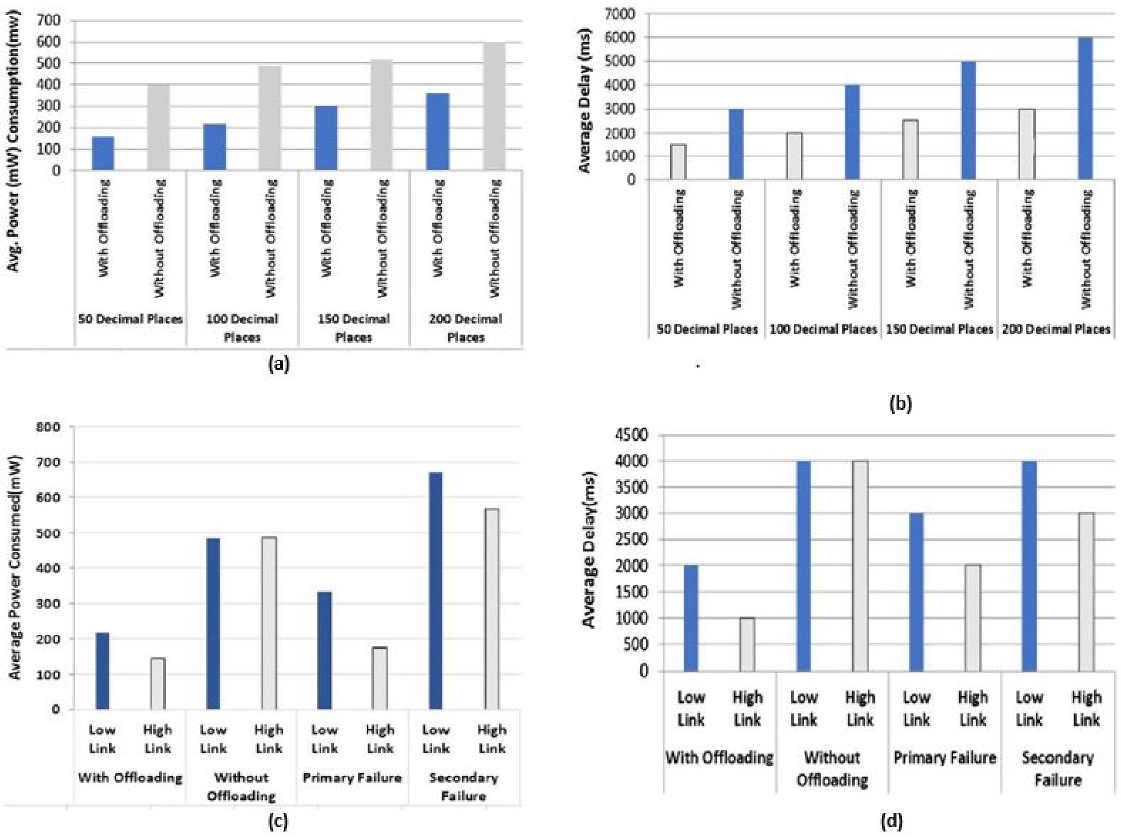


Figure 5.6: Operational inspection of πcalculator when executed fault free and faulty in offloading and in absence of offloading scenario; (a) average power consumed by different components of πcalculator for failure free execution; (b) average delay for different precision of result in πcalculator for failure free execution; (c) average power consumption for varying link speed (leading to omission failure) and crash failure conditions; (d) average delay for varying link speed (leading to omission failure) and crash failure conditions

address" and "port no" to make a connection with the surrogate in runtime.

3. Offloading in presence of fault-tolerance: The proposed fault-tolerance approach is used in this step where the a "Network Discovery" method is made run to find the list of available surrogates. The most appropriate surrogate is chosen from the list and the task is sent.

To evaluate the framework, a simple Java application, πcalculator is being used. This application gives a specific value of π to the mentioned precision, where the user specifies to which decimal places the value should be computed and accordingly enters the value to the πcalculator. The surrogates extract checkpoint routinely after every 50 iterations of calculations. Serializable interface is implemented at surrogates

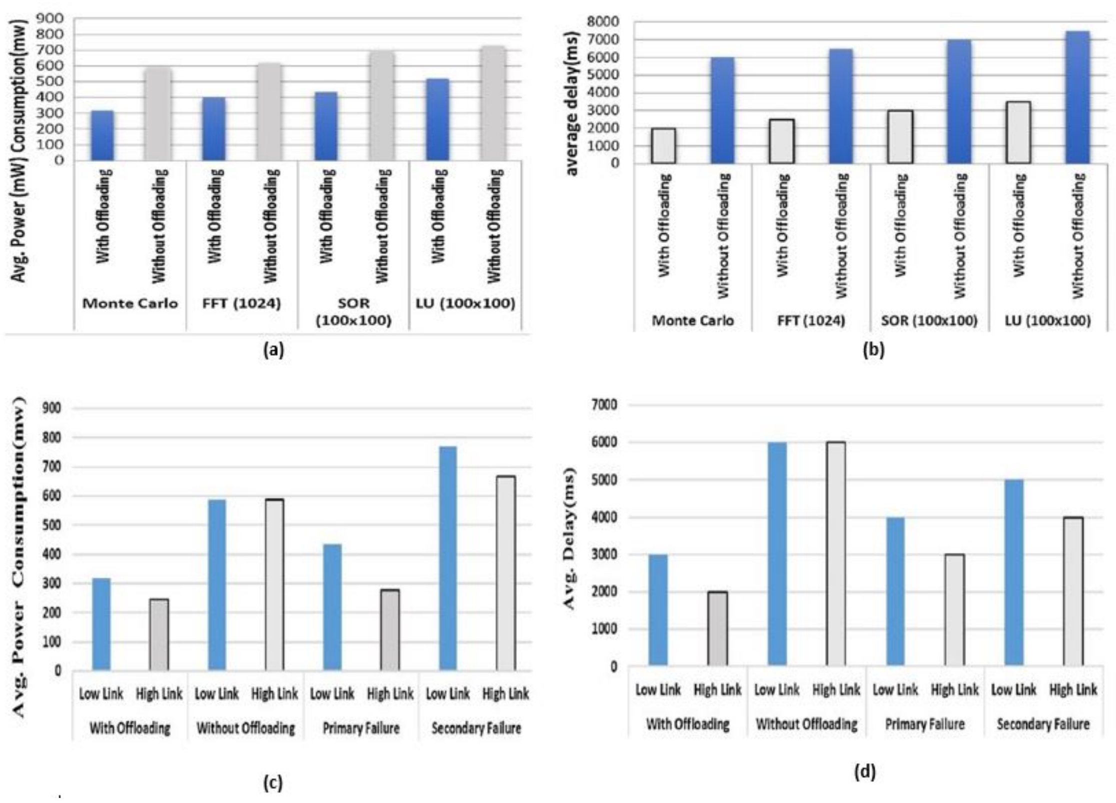


Figure 5.7: Operational inspection of Scimark subject to executed fault free and faulty in offloading and in absence of offloading scenario; (a) average power consumed by different components of Scimark for failure free execution; (b) average delay for different components of Scimark for failure free execution; (c) average power consumption for varying link speed (leading to omission failure) and crash failure conditions; d average delay for varying link speed (leading to omission failure) and crash failure conditions

to form checkpoints which also stores the value of the loop variable. The secondary surrogate receives the file where the serialised object is written. The secondary surrogate starts execution of the task after serializing the last checkpoint object as a crash failure is forcibly injected by throwing an Exception at the primary surrogate. A well known benchmark for computation intensive application, Scimark benchmark suite[1], is utilized for accessing the performance of the system. At the offloadee devices, Android plugin of Scimark is installed and at the surrogates, Windows plugin is installed so that offloading applications can be executed. For this work, the following Scimark suite are examined:

- Monte Carlo integration (MC): By calculating the integral quarter circle $y =$

[1]https://math.nist.gov/scimark2/about.html

$\sqrt{1-x^2}$ on [0,1] the approximate value of π is obtained. Synchronized function calls, random-number generators etc. is utilized by the algorithm.

- Fast Fourier Transform (FFT): One-dimensional forward transform of 4K complex number is performed with utilization of complex arithmetic, shuffling etc.
- Jacobi Successive Over-relaxation (SOR): Typical access pattern is employed on a 100 x 100 grid, in finite differential applications. Basic "grid averaging" memory pattern is being used by the algorithm.
- Dense LU matrix factorization (LU): By utilising partial pivoting on a dense 100 x 100 matrix, the LU factorization is performed. It utilizes linear algebra and dense matrix operations.

5.3 Experimental setup

In this chapter, client-server based paradigm is utilized for fault-tolerant framework, where the offloadee is considered to be the client/ mobile device having limited computation capability and is supposed to be resource constrained whereas the surrogates are resource rich and having high computation capabilities and are interested to compute and/or store information or result on behalf of the offloadee. Surrogates are generally available through local LAN (e.g., smart homes, offices or commercial hot-spot environment) or internet and can be desktop PC, or laptops. Here, connection is done through IEEE 802.11n WLAN standard and the region of implementation is considered as the university campus and the offloadee is the smart hand held device and the surrogates are the laptops. For implementing this work, four HPProbook4550 series Laptops with 2,10-GHz i3 processor with 1GB of RAM is used as surrogates. the number of offloadee taken here is four (with specification Samsung Galaxy GT-P5100 Tab running Android v. 4.1.2, Samsung Galaxy Tab10 having Android v. 4, Micromax Yureka with Android v. 5.1.1 and Samsung

Galaxy Tab E with Android version 4.4). IEEE 802.11 WLAN is used to connect the surrogates and the offloadees. The code for offloadee is written in Android Studio and surrogate is written Java making it flexible to use it in any platform. LAN connection with varying speed (between 65-75 Mbps) is used in this work.

5.4 Results and discussions

In this section, experimental results are evaluated and analysed. Two use cases have been taken into account: (i) tailor made application such as πcalculator and (ii) Application suite Scimark Benchmark which consists of computational kernels MC, FFT, SOR and LU. This work considers two metric for evaluation, i.e., energy consumption and response time. For different setup of parametric values, Table 2. describes the comparisons of the execution time. The request response ratio is considered to be of a non zero value when the offloadee has a familiar neighbourhood.

During the time interval where the neighbourhood is a stable one, it can be noticed that the success rate is maximum when network conditions and success rate are assigned equal weights (0.5) following Eq 5.6. This parametric combination is kept as the default value, while performing the remaining experiments. The amount of energy utilized by this approach is calculated by a software, PowerTutor[[116], [117]] which assists in measuring the power consumed by the various applications/ components such as CPU, network interface, display by an application, that are executing on the Android device at that particular time. Fig 5.6. depicts the comparative analysis of energy consumption of πcalculator in three sets of conditions i.e., local execution, with offloading and with fault-tolerant offloading, where the X-axis depicts the accuracy of calculation of π (i.e., to number of decimal places) and Y-axis depicts the usage of resources. Fig 5.6(c) describes the power consumption profile of offloadee for different applications, when there are primary and/or secondary surrogate failure for varying link speeds. The repercussion of crash and

Table 5.2: Execution time(ms) comparisons ofπ-Calculator and Scimark

	Success Rate (SR)			
BenchMark Application	$\alpha = 1, \beta = 0,$ $\delta t1 = 0,$ $\delta t2 = 0,$ $(res^s_n/req^s_n) = 0.5$	$\alpha = 0.5, \beta = 0.5,$ $\delta t1 = 100,$ $\delta t2 = 300,$ $(res^s_n/req^s_n) = 1$	$\alpha = 0.2, \beta = 0.8,$ $\delta t1 = 100,$ $\delta t2 = 300,$ $(res^s_n/req^s_n) = 0.33$	$\alpha = 0, \beta = 1,$ $\delta t1 = 100,$ $\delta t2 = 300,$ $(res^s_n/req^s_n) = 0$
π**Calculator**	0.5	0.55	0.10667	0.0333
Scimark	0.5	0.575	0.14667	0.1

omission failure can be explained by the said situation on the performance of the system.

It can be presumed from Figs. 5.6(c) and 5.6(d) that the power consumption and the response time can be conspicuously diminished when implementing offloading in comparison to local execution, while it is noticed that there is a slight increase in values of the parameters when fault-tolerant mechanism is implemented additionally. However, it can be safely be asserted that even with the implementation of fault tolerance in offloading, it is still advantageous with respect to local execution for most of the cases.

Fig. 5.7 synopsizes the results of Scimark benchmark applications. When compared with other kernels of Scimark, LU has longer response time, but is found from Fig. 5.7(a) and 5.7(b) that offloading is advantageous for all the kernels equally. Here, omission failure is introduced in the system, by lowering the average link speed which increases the possibility of omission failure. Though, slow network does not have any effect on the local execution but it hampers the offloading system when the task is offloaded to the surrogate servers as the communication gets noisy. Nevertheless, less response time and less energy is consumed when offloading the results in such low speed network conditions in both cases of πcalculator (from Figs. 5.6(c), 5.6(d)) and Scimark (from Figs. 5.7(c) and 5.7(d)). In case of primary surrogate failure, the secondary surrogate resumes the tasks from the checkpoint that is last saved, but in case of secondary surrogate failure, local execution of the tasks in the Android device is the only alternative. The proposed approach performs better than

[81] when average power consumption or average delay of network parameters are considered, even when the link condition is low. From Fig.5.7(c) and 5.6(c), it can be observed that a comparison is performed between Scimark and πcalculator, storing the checkpoints at offloadee, and in case of failure of surrogate, local execution is performed as in [81]. The remaining of the task is taken over from the last saved checkpoint by the secondary surrogate, when crash failure occurs at primary surrogate, and this failure is noticed only after the timeout at secondary surrogate. Yet, the system performs faster and the fault tolerance overhead is minimal as compared to local execution. Here, the surrogates at the edge where the offloading is to be done and the network connections are assumed to be secure. Also, with time, low values of S_k^{SR} will be assigned to malicious surrogates which will stop the offloadee in selecting the malicious surrogate in case of failure.

5.5 Summary

In this chapter, at first, different causes of failures in offloading is identified and classified along with its repercussion on offloading system which have computation capabilities. Crash failure and omission failure predominantly impacts the offloading system. Additionally, security can also cause failure in the offloading system. Next, an innovative fault-tolerant framework is put forward which takes control of the crash failure at the surrogates, and by redundancy and hashing method it handles the omission and transient failures. Risk of failure is reduced by using trusted surrogates which will in turn prevent the occurrence of faults. A combination of timeout and hash mechanisms aids in detection of failures where checkpointing helps in attaining fault tolerance. The overhead of keeping checkpoints in offloadee can be minimized by migrating the checkpoints to the secondary surrogate.

By utilising smart handheld devices based on Android OS, the framework has been implemented in a platform independent manner where both customized application (for eg., πcalculator and Scimark benchmark application suite are utilized. The performance of the fault-tolerant framework is analysed by considering the parameters resource utilization and energy consumption. Even with the presence of failures, it is observed that the framework preserves the benefits of offloading to the edge servers. But if the offloading can be done to the edge servers which are nearer to the mobile devices then these types of faults can be further reduced as these edge servers are located near to the mobile devices. Offloading the tasks to the fittest edge servers can be done by optimization of multiple parameters. Hence, meta-heuristics can be applied to decide about a suitable strategy for task offloading. These are discussed in the next chapter.

List of Publications:

Journal:

1. Chowdhury, C., Roy, S., **Ray, A.** and Deb, S.K., 2020. A fault-tolerant approach to alleviate failures in offloading systems. Wireless Personal Communications, 110(2), pp.1033-1055.

Chapter 6

Computation and Load Balancing Approaches to Allocate Tasks in Edge Computing

Edge computing [118] is a distributed computational paradigm that has been responsible to broaden the cloud services to the edges where the edges form the logical layer positioned as close to the data sources as possible, and it forms the inter-medial layer joining the edge servers to the cloud servers. There is still a few limitations of edge computing. If single standalone edges are considered that are geographically distributed and servicing a number of users, the edges are not as efficient as cloud servers when a comparison is made with respect to capacity of the resources and the computational power [118]. In addition to these issues, these individual standalone edge nodes do not share information with the neighbouring edge nodes located in nearby areas since the edge nodes are not aware of other edge nodes due to the fact that these nodes are geographically distributed. This results in under-utilization of the resources. So, an efficient optimization of resource utilization at less time is very difficult to attain. Therefore, the QoS deteriorates due to the lengthy execution and waiting time and interruptions of services. Thus, this chapter discusses one such strategy related to load balancing at the edge servers, objectives and contributions as follows.

6.1 Load allocation utilising SA

Problem description The standalone edge servers distributed geographically over an area can not use the resources in an optimized way. Hence, a solution is needed to aggregate these standalone edge servers to utilize these edge nodes efficiently while guaranteeing low latency and reducing the energy utilization.

Objective The objective is to offload tasks that are computationally intensive to the edge servers which are closest to the mobile devices by the Edge federation, such that the tasks are distributed in a more balanced and cost-effective manner utilising SA.

Contribution A modified edge computing framework- Edge federation is proposed to provide a transparent and seamless service to the end user, which will reduce the latency and optimize the energy utilization and distribute the tasks in balanced manner to the fittest edge servers by the Edge federation, when the nearest edge server fails to provide the service. Simulated Annealing has been applied here for the purpose.

The rest of this chapter is organized as follows: The Section 6.2 explains in details the architecture of the proposed approach, Section 6.3 explains the details of the proposed approach, Section 6.4 states the experimental setup and Section 6.5 discusses the results utilising SA and Section 6.6 summarises the whole chapter.

To explain the Edge federation framework, a real life scenario is examined Fig. 6.1, where a building in university campus is considered as the total available space for wireless communication, and each floor consists of classrooms and laboratories and has e number of edge servers. It is evident that most of the people in the university building carry smart mobile devices which are equipped with a lot of

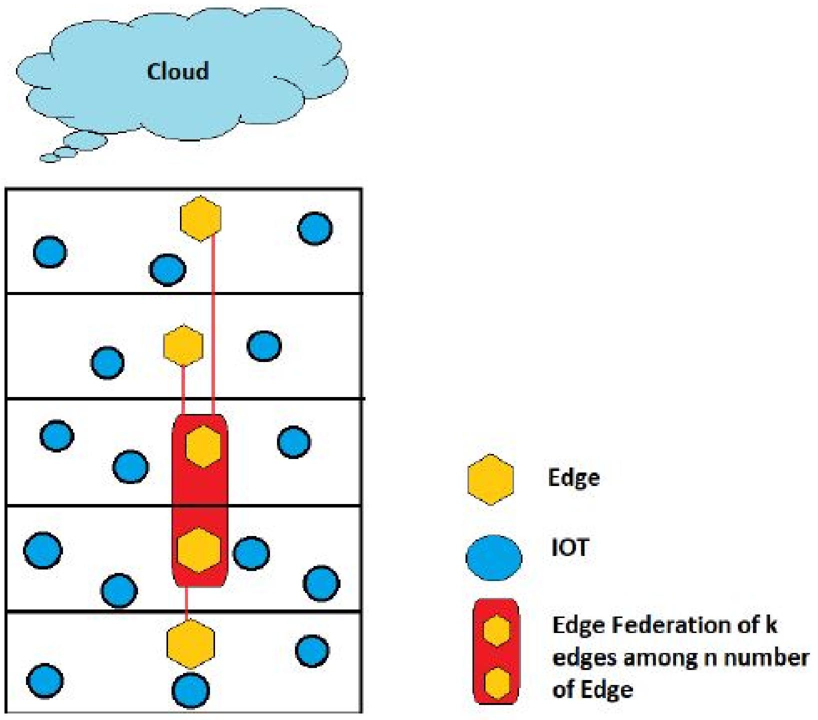


Figure 6.1: Pictorial representation of a real life scenario of the proposed Edge federation system

sensors. The computation intensive tasks/data from these IoT devices are offloaded to its closest edge server. Among the various edge servers, servers with high processing power and resources are grouped in the Edge federation within the building or among various buildings within the university. If the nearest edge server is not capable of performing the task at that given time, then that task is offloaded to the Edge federation. The Edge federation generally takes the decision of sending the tasks to the fittest edge servers, which has the capability of completing that tasks. In case the Edge federation is unable to get the tasks done only then the tasks are sent off to the cloud server for completing the task.

6.2 Architecture of Edge federation

In Edge federation framework, a four-tier architecture is proposed. The Edge federation is an extra layer between the edge servers and the cloud servers. In Fig. 6.2, it

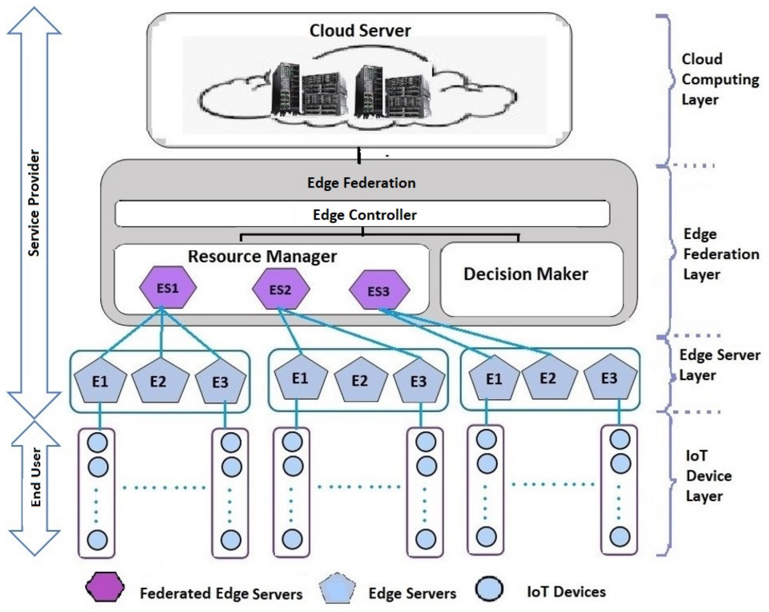


Figure 6.2: Architecture of the proposed Edge federation system

is seen that the end users are in the lowest layers, that have various IoT devices for different applications. Each of the IoT devices, searches for its nearest edge server for offloading the task. A number of edge servers near vicinity forms a group and remain connected to its nearest Edge Federation. This Edge Federation is formed by group of edge servers (in this example it is ES1, ES2, ES3). Edge federation have all the information about all other edge servers, of the whole building. The Edge federation comprises of collection of k edge servers out of e edge servers i.e, Edge federation will be a collection of k numbers of edge servers present in a geographic location, where $k <= e$. The main functional unit of Edge federation is the edge controller (similar to the concept of edge orchestrator) [119], [120]. It has two components namely:

1. Resource Manager: It is responsible for managing the various physical resources of the edge servers that is connected to the Edge federation. It keeps a record of the load at each edge server connected to it along with the information of the tasks that are being processed in the edge servers.

2. Decision maker: It is responsible for taking decision on behalf of the Edge

federation. It distributes the tasks to the edge servers, based on the load at that edge server and the computational power of the edge servers. It also keeps an account of the tasks that are being executed at the edge servers. When there is any failure in task execution, the Edge federation takes the decision of either allotting the tasks to another edge server or to the cloud server.

Allocating the task from the Edge federation to the appropriate edge server is performed by implementing the optimization algorithm - SA.

6.3 Task allocation by Edge federation by implementing SA

Edge federation accomplishes the tasks allocation by implementing a heuristic method- Simulated Annealing for allocation of the tasks to the fittest edge servers so that the task execution time is reduced significantly. The tasks allocation done by the resource manager in Edge federation is an NP-complete problem. In the following segment, first, the algorithm is described and then the mapping of the algorithm is explained i.e., how the algorithm is utilized for solving the tasks allocation problem in Edge Federation.

i) Some basic concepts of SA algorithm:

Globally lowest energy point in an energy distribution environment [121], [122] can be obtained by the SA [123] optimization algorithm. The algorithm was developed when scientists noticed the nature in which a cooled molten metal gradually may form a regular crystalline structure. So, a metal is heated up till it reaches an annealing temperature, after which it is cooled slowly such that the metal can be changed according to its desired structure thereby minimising the potential energy of the mass. The algorithm introduces random changes/ jumps to explore a possible new and better solution space while avoiding local optima. As the algorithm proceeds, these random changes are done in a more controlled and decreasing manner.

The parameters of SA- temperature and energy have been mapped to the task allocation problem. Energy minimization is the objective function and the temperature is used for efficient searching of the solution space avoiding local minima. For an ascending move in the system, temperature (T) is the acceptance probability and the range to which the temperature rises is denoted by delta (Δ). The system either shifts to a reduced energy state with the new solution which results in better fitness. Otherwise, it shifts to the new solution with a probability $P = exp(-\Delta E/kT)$, to a higher energy state, where ΔE is the increase in energy.

Algorithm 3 explains the task allocation implementing SA. Initial solution S_s is randomly selected at the beginning of the algorithm and accordingly, the algorithm calculates the energy/cost $S_s.Energy$ for the present solution S_s. An initial temperature T is set, and then neighbouring solution S_{next} and corresponding energy/cost $S_{next}.Energy$ is computed in reference to the present solution S_s. If the newly calculated energy/cost of any neighbouring solution is less than the existing present cost $S_s.Energy$, then the new solution S_{next} is accepted, where $S_{next}.Energy$ is the energy/cost of the newly obtained solution. Else, the neighbouring solution is also adopted as the new energy/cost solution, depending on the value of Δ, where $\Delta =$ $S_s.Energy$ - $S_{next}.Energy$ and exp(-Δ/T) is the probability function used as the deciding factor to accept the new energy/cost solution. On arriving to the thermal equilibrium at temperature (T) for the system, this temperature is reduced by a cooling factor α, and the inner iteration loop is increased by an increment factor β, and the algorithm carries on till it reaches another new thermal equilibrium point with the new diminished temperature. The algorithm stops if it has reached T_{min}, minimum temperature, or the maximum number of iterations. This is stated as the *stopping condition.* When the algorithm is implemented for finding the optimized solution to a given problem, a number of factors need to be looked into namely, the

Algorithm 3 Task Allocation Implementing Simulated Annealing

```
1: Input: Tasks (Kbps, in million instructions, latency), edge servers profile $S.Profile$
   (MIPS, max energy $S_s.EC$)(where $S$ is the set of edge servers in Edge federation)
2: Output: Solution servers $S_s$ (Edge servers to which the tasks will be offload)
3: Initial temp $T$, $\alpha = 0.9$ (considered for implementation) ($temp\_reducing\_factor$ $\alpha$
   $<1$); $\beta = 1.05$ (considered for implementation) ($chain\_increasing\_factor$ $\beta >1$);
4: Select: Random initial solution $S_s$ from $S$ (set of edge server), random initial chain
   variable $n_{rep}$;
5: $S_s.Energy = Energy\_Cost$ ($S_s.Profile$, Task);
6: Repeat
7: for $v = 0$; $v <= n_{rep}$; $v = v + 1$ do
8:     $S_{next} = S_s$;                                   ▷ Neighbouring solution selection
9:     $S_{next}.Energy = Energy\_Cost(S_{next}.Profile, Task)$    ▷ Calculate the energy of
   the neighbouring selected server
10:    $\Delta = S_{next}.Energy$ - $S_s.Energy$     ▷ difference of energy of current server and
   neighbouring server
11:    if $\Delta < 0$ then
12:        $S_s = S_{next}$;      ▷ neighbouring server selected $S_s.Energy = S_{next}.Energy$;
13:    else
14:        $x = rand(0, 1)$                                ▷ a random number is selected
15:        if $e^{-\Delta/T} > x$ then
16:            $S_s = S_{next}$;                           ▷ neighbouring server selected
17:            $S_s.Energy = S_{next}.Energy$;
18:        end if
19:    end if
20: end for
21: Set $T = \alpha \times T$
22: $n_{rep} = \beta \times n_{rep}$
23: Until Stopping condition
24: Return Solution set of servers $S_s$ (Edge servers to which the tasks will offload)
    $S_s.Energy$ (the energy consumed)
```

selection of the energy function, the cooling schedule, the structure of the neighborhood, and annealing parameters. All these parameters are critical as they are deciding factors for the standard of the solution obtained and the speed at which the solution is obtained.

When the algorithm is implemented for finding the optimized solution to a given problem, a number of concerns needs to be taken into account. The selection of the energy function, the cooling schedule, the structure of the neighbourhood, the cooling factor, the increasing factor and the algorithm stopping criteria. All these parameters are critical as it is a deciding factor for the quality of the solution obtained and the speed at which the solution is obtained.

1. Energy function

   The whole purpose of the algorithm is to reduce the energy that is being used in finding the solution. The energy function is calculated as follows:

   $$S_s.EnergyCost = \sum_{i=1}^{|S|} \frac{S_i.power \times S_i.Tasklength}{S_i.MIPS}$$

   The energy is calculated in watt-hour and $S_s.EnergyCost$ [124] is the energy that is utilized by the edge servers where the tasks are initially allocated, $S_i.Tasklength$ indicates the tasks allotted to S_i.

2. Neighborhood structure

   The set of moves that are implemented to the present solution to obtain a new solution i.e., to shift from one equilibrium point or solution (S_s) to another (S_{next}) in the nearby solution space, is defined as neighbourhood. In this work, the neighbourhood solution is obtained by first selecting a random solution S_s and allocating the task to an edge server, and then based on Δ, where $\Delta=$ $S_s.Energy$ - $S_{next}.Energy$, the next equilibrium point S_{next} (another task allocation schedule) is obtained.

3. Cooling schedule

   The feature that can be changed and controlled in SA is the temperature T. The strategy that is used to decrease the temperature of the process till the thermal equilibrium state is reached is defined as the cooling schedule. In this work, a cooling schedule which is geometric in nature is taken into consideration where $T= \alpha \times T$, where $\alpha(< 1)$ is a constant known as the cooling rate.

This cooling schedule is controlled by the number of iterations in the inner loop n_{rep} and the cooling rate α. The number of iterations in the inner loop at every temperature, T, is modified according to $n_{rep} = \beta \times n_{rep}$, where β is a constant greater than 1.

4. Annealing parameters

   - Cooling (reducing) rate α: It is defined as the rate at which the temperature (time to complete the task) is decreased. It is found in [122] and [125] that if the cooling rate α is between 0.9 and 0.90 then there are higher chances of success. Here, the rate of cooling is selected to be $\alpha =$ 0.90.
   - Increasing rate β: As the temperature T is reduced, the rate at which the number of iterations in the inner loop increases, is defined as the increasing factor β. In this work, the value is chosen to be β= 1.05. It is decided empirically.
   - Initial temperature T: This is the starting temperature of the system and it takes the decision of the probability to accept a worse solution. Initially, the time is set to a fairly high value to capture a large horizon of the solution space, then slowly reducing this value to obtain the optimal solution.

Fig. 6.3 describes the workflow of the proposed system. When a task is offloaded to the nearest edge server, the edge server checks if it has the capacity to perform the task, if it cannot perform the task, the edge server sends the tasks to the Edge federation. The Edge federation allocates the task to the fittest edge server and after the task is completed, that edge server sends back the completed task back to the mobile devices which has initiated the task.

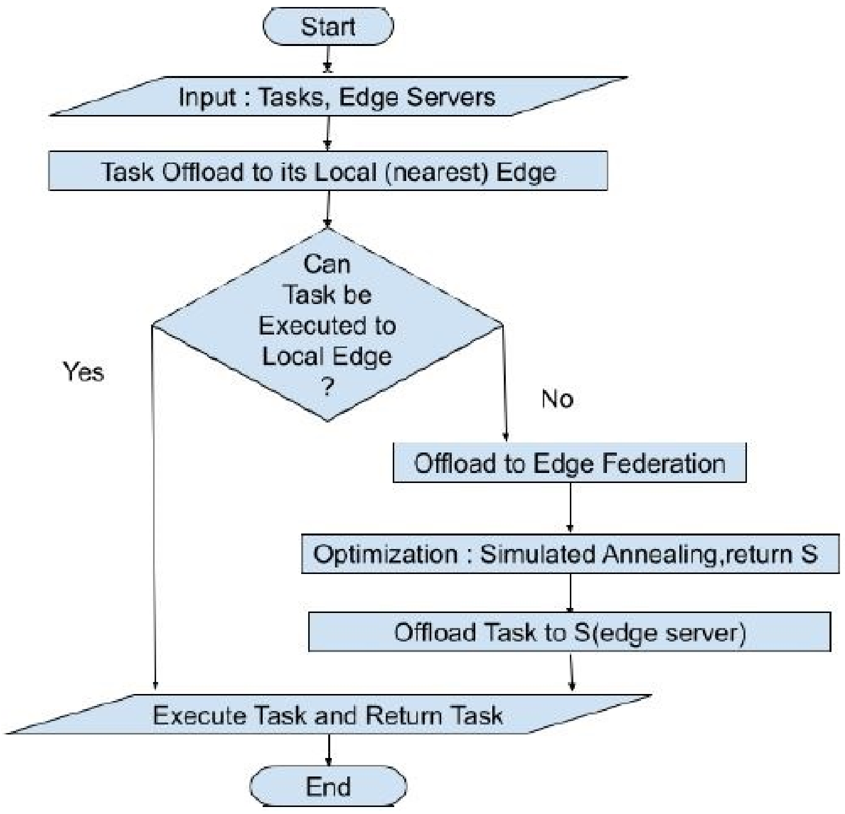

Start
Input : Tasks, Edge Servers
Task Offload to its Local (nearest) Edge
Can Task be Executed to Local Edge ?
Yes
No
Offload to Edge Federation
Optimization : Simulated Annealing,return S
Offload Task to S(edge server)
Execute Task and Return Task
End

Figure 6.3: Workflow of the proposed system

6.4 Experimental setup

A prototype of the proposed work is simulated using the PureEdgeSim [124] simulator. The system uses Windows 10 as a platform. PureEdgeSim is an object-oriented framework built on CloudSim Plus. It is the framework within which the device-edge-cloud architectures could be effectively simulated. SA is implemented here for task allocation. In the proposed work, three types of tasks have been considered for offloading namely, a) augmented reality, b) healthcare, and c) high computational task. It is assumed that the augmented reality task will need the least and the high computational task will need the most computational requirements. The healthcare system will need average computational requirements. The algorithm is simulated for a large number of tasks generated randomly. Here, 2400 tasks are considered. These tasks have varying levels of computational requirements of the servers, having medium to high computational capabilities, and are allocated within the distributed environment. Five active edge servers are taken into consideration where they are placed at different locations within an area of 100 $sq.m.$ The edge servers are placed in such a way that they are at least 10 m apart from each other. After the simulation, all the outputs are saved as a text file (.txt), excel (.csv) file, and the graphs in PNG format.

6.5 Results and discussions

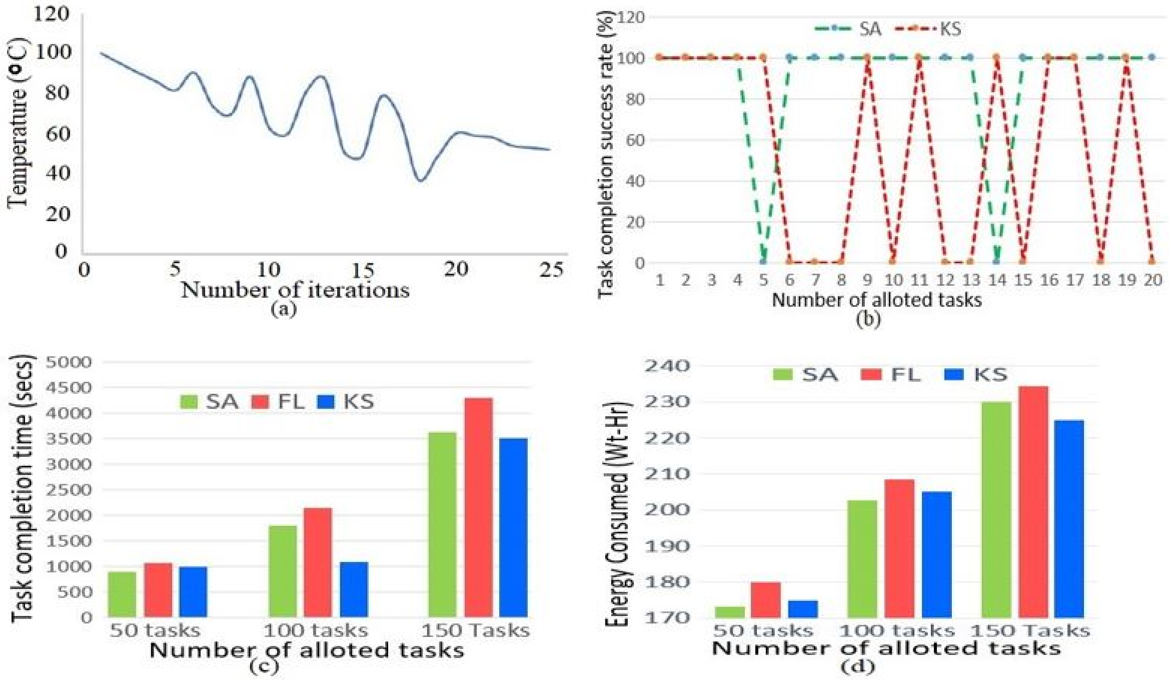


Figure 6.4: (a)Total Number of iterations to reach stable state (b) Success rate of allocated tasks of SA & KS (c) Task completion time implementing SA, FL & KS (d) Energy Consumed implementing SA, FL & KS

The proposed algorithm is simulated with a sizeable number of tasks that are generated randomly and are allocated in a distributed system. In Fig. 6.4(a), the plotted graph shows the rate at which the system attains an equilibrium state by slowly reducing the temperature.As the algorithm progresses, a steady state is obtained when the number of iterations is more than 20. The cooling factor is kept between 0.90 and 0.95.

An experiment is conducted to compare the task allocation success rate of the proposed algorithm with the knapsack algorithm as shown in Fig. 6.4(b). Tasks that cannot be executed by the edge server due to issues of mobility are considered failed tasks denoted as 0% success and tasks which are successfully executed after offloading are considered as success, denoted as 100% success. Five edge servers are taken into consideration with 20 tasks. Task 5 and Task 14 failed to be allocated due to a time delay in executing the tasks in the allocated edge servers and the rest tasks are successfully assigned and executed when SA is implemented, but task number 6, 7, 8, 10, 12, 13, 15, 18 and 20 failed to successfully complete when implemented by KS. From the graph, it can be concluded that 18 out of 20 tasks are successfully executed by the edge servers implementing SA while 11 tasks are successfully executed by edge servers with KS. Hence, accuracy of the success rate by the proposed work is 90%. When the standard FL was used instead of SA, it was found that, the success rate was nearly 43.98% due to high delay and the total task execution time was 1074.25 seconds.

A comparative analysis of the proposed SA, FL, and KS has been performed in terms of the task completion time as shown in Fig. 6.4 (c). Task completion is referred to as the process of offloading the tasks to the edge servers from the IoT devices, executing the tasks on edge servers, and returning the executed tasks with suitable output to the IoT devices. The task completion time for allotted tasks, initially is the lowest for the proposed approach implementing SA while FL and KS took more time to complete the allotted tasks. But with time, KS algorithm and SA shows comparable performance. However, task failure rate kept on increasing up to 45% with the KS algorithm.

Fig. 6.4 (d) presents the resulting energy expenditure of the algorithms. It has been found that the consumption of energy was initially lowest for the proposed work. However, as the number of tasks increased, KS algorithm performed comparably with the proposed work though the proposed work resulted in more number of tasks to be allocated and completed. From the simulation outputs, an inference can be drawn that implementing the SA, the algorithm works most efficiently with 15 to 25 edge servers and 100 to 350 tasks. Therefore, using SA as an optimization algorithm provides better performance with respect to task execution and successful completion of the tasks, utilizing energy in an optimized manner.

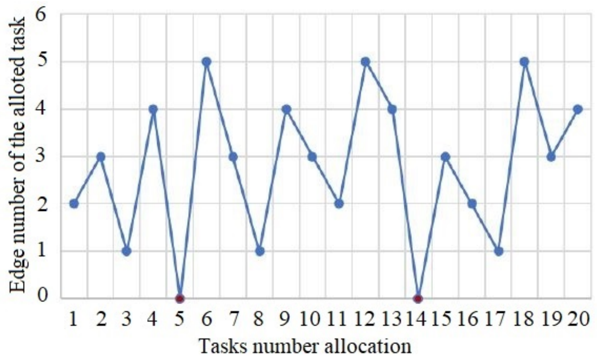


Figure 6.5: Task assignment to edge servers

Fig. 6.5 shows the tasks allocation to the various edge servers. Among the five edge servers 20 tasks are allocated where, task 5 and 14 failed to be allocated. From the figure it is observed that server 3 is allocated maximum number of tasks i.e. tasks 2, 7, 10, 15, 19. Task 5 and 14 are not alloted any server due to delay in sending tasks by IoT devices to edge server.

6.6 Summary

In this chapter, an energy-efficient task allocation based on SA algorithm is described, and an integrated service provisioning model named, Edge federation is discussed. A four-tier architecture comprising edge servers, IoT devices, and Edge federation is designed so that energy consumption of the system can be reduced by offloading the tasks to the most suitable edge servers and not offloading the task to a cloud server. An intermediate layer, the Edge federation, takes over the responsibility of allocating the tasks to the fittest edge servers near its vicinity. To swiftly obtain a solution of the task allocation problem, a near-optimal solution is found by implementing an allocation method based on the SA algorithm. When simulated on a sizeable number of randomly generated instances, these instances were used to assess the algorithm's performance. According to the experimentation, the proposed SA-based task offloading by Edge federation results in better performance as compared to state-of-the-art techniques.

The next chapter, is the concluding chapter that concludes and summarizes the total thesis. Along with it, the future scope of this research are discussed in the next chapter.

List of Publications:

Conference:

1. **Ray, A.**, Prasad, R., Chowdhury, C., and Roy, S., 2023. Energy Efficient Task Execution Through Edge Federation Utilizing Simulated Annealing. In the proceedings of International Conference on Security, Surveillance and Artificial Intelligence, ICSSAI 2023. Part 3, Ch 25, pp.(247-257).

Chapter 7

Conclusion and Future Scope

Mobile crowd-sensing and sourcing are the latest paradigms in the field of IoT and smart sensing applications. As the resources in these devices are limited hence there is an urgent need to optimize the utilization of resources at the same time maintaining the quality of services (QoS). A number of approaches has been proposed to address these issues. The proposed frameworks are directed to utilize the energy of the mobile devices in an optimized manner and hence motivate the mobile crowd to participate in crowd-sensing tasks. This chapter concludes with a review of the work presented in this thesis. The overall significance of the results are summarized. A few directions for future work are also suggested.

7.1 Summary of the thesis

The comprehensive researches of this thesis is summarized as follows:

1. An efficient framework combines the effectiveness of mobile crowd-sensing is discussed in Chapter 3 by utilising the resources in an optimized fashion as well as providing certain rewards on performing the tasks.

2. As the smart handheld devices are participating in sensing or performing certain tasks, in order to keep the mobile devices active for longer duration of

time, a strategy is formulated based on MDP in Chapter 4. The strategy helps the mobile devices to decide when to participate in crowd-sensing and when to stop sensing of data based on the devices' current load and recharging conditions. The strategy is implemented in a indoor localisation scenario. It has been observed that the performance of the devices are more efficient than some benchmark strategies for different load conditions of the mobile devices.

3. In order to handle time-sensitive data or tasks which can drain a lot of resources of the mobile devices, often the data/ tasks are offloaded to the backend servers, such as edges. While offloading, different types of failures are encountered often, which are classified in Chapter 5. An innovative fault tolerant framework is discussed which helps in handling crash, omission and transient failures and checkpointing is used to make the system one degree fault tolerant. So, even with the presence of faults, it is observed that the framework preserves the benefits of offloading to the edges.

4. It has been found that offloading the tasks/ data to the clouds has certain issues which can be reduced when it is offloaded to the edge servers that are nearer to the mobile devices. In Chapter 6, optimization techniques, namely, SA has been utilized to optimally schedule the tasks at the edge servers and it has been observed that SA has performed better than different benchmark techniques in optimising the resource utilization and distribution of the offloadable tasks to the edge servers.

5. In this thesis, programming languages such as R, Java have been utilized along with Android studio IDE for developing the Android apps. PureEdgeSim simulator is being used to simulate the edge servers and compare the work with different techniques.

7.2 Summary of the contributions

This section condenses the chapter wise contribution of the research.

In Chapter 3, a novel framework is proposed which amalgamates the benefits of mobile crowd-sensing and crowd sourcing in solving a task locally without offloading the task to the backend server. This is done in much reduced time. The results thus obtained can be gathered in both ways, i.e., online, when there is internet connectivity, or offline when there is no internet, by utilising the intelligence of the crowd, for e.g., the crowd intelligence is utilized to obtain information of a route, when the user is not connected over the the internet.

An energy efficient strategy based on MDP where a smart handheld device takes the decision of participation in crowd-sensing is proposed in Chapter 4. Here the proposed strategy considers possible device conditions prior to deployment depending on various factors namely, remaining power of the device, the load on the device at that point of time, battery recharging and the crowdsensing incentive. The output thus obtained is an energy efficient strategy that helps the mobile devices to decide when to crowd-sense and when not to, based on the various conditions of the devices. For e.g., the mobile device will crowd-sense irrespective of the load conditions in the device when there is probability of the mobile device to recharge in the next time slot. This strategy based on MDP is implemented in an Android application to facilitate crowd-sensing in real time.

A efficient framework is proposed in Chapter 5 which discovers and classifies the various faults and failures that are encountered when tasks/ data are being offloaded to the resource rich, computation intensive backend servers. This framework not only helps in detecting and classifying faults when it occurs but also helps in tolerating one degree failure. This condition happens when the primary surrogate fails to perform the offloadable task. The failure is tolerated by the help of checkpointing. The secondary surrogate resumes the task from the checkpoint that was last saved and resumes the task from that checkpoint and thus the secondary surrogates can help in preventing crash failures.

In Chapter 6, offloadable tasks were distributed to back end edge servers by utilising an optimization algorithm, SA in a balanced way so that the resources are utilized in optimized manner. Here, a modified edge computing framework-Edge federation is proposed to provide a transparent and seamless service to the end user, which will reduce the latency and allocating the tasks fairly to the fittest edge servers by the Edge federation, when the nearest edge server fails to provide the service, is done based on the principle of Simulated Annealing. The proposed work is simulated in PureEdgeSim [124]. The results were compared with Knapsack (KS) algorithm and Fuzzy logic (FL) and it was found that the energy consumption in SA were comparable to the two benchmark algorithms, namely KS and FL. Even the percentage of successful completion of the tasks are better in SA than the rest two algorithms.

7.3 Future scope

In this thesis, a number of frameworks and optimization techniques have been proposed for utilising the resources of the mobile devices efficiently. However, there are quite a number of areas where further researches need to be done to study the performances of the systems in those given conditions. In this section, the directions in which this research can be extended are discussed.

In Chapter 3, the proposed approach considered taking aid of human intelligence while making offline decisions, however, the various patterns of human behaviour in case of offline scenario are not taken into consideration. This human behavioural pattern can be delved into for the future research areas.

In Chapter 4, the strategy based on MDP is implemented, where a mobile device takes the decision of lending itself to crowd-sensing based on the device specific factors such as, the current load on the device [4]. However, the factors can be extended by taking into consideration network related issues such as, the available bandwidth of the device at that time instance. Also, congestion of the network can be a deciding factor while taking the decision to crowd-sense.

In Chapter 5, the proposed framework attempts to tolerate fault of first degree by introducing secondary surrogate and offloading the checkpoints to the secondary surrogates. The framework takes control of the crash failure at the surrogates. It also handles the omission and transient failures, by applying redundancy and hashing mechanisms. Risk of failure is reduced by using trusted surrogates which will in turn prevent the occurrence of faults. A combination of timeout and hash mechanisms aids in detection of failures where checkpointing helps in attaining fault tolerance. The overhead of keeping checkpoints in offloadee can be minimized by migrating the checkpoints to the secondary surrogate. Further research needs to be done to study the effects of introducing more surrogates and observe if the degree of failures can be reduced.

In Chapter 6, load allocation to the edge servers are done based on SA optimization technique that has been simulated by PureEdgeSim. A modified edge computing framework- Edge federation is proposed to provide a transparent and seamless service to the end users, which will reduce the latency and allocate the tasks fairly to the fittest edge servers. When the nearest edge server fails to provide the service, Edge federation allocates it based on the principle of Simulated Annealing.

While implementing SA, geometric cooling schedule is considered. It has been found that by using constant thermodynamic speed annealing schedule for cooling schedule, better low-energy solutions to problems can be obtained [126]. Hence, in the extension of this work, a plan is in progress to implement a thermodynamic speed annealing schedule and perform a comparative analysis among the various cooling schedules of SA. Another extension of this work is by implementing BPSO, NBPSO [127] (New Binary Particle Swarm Optimization), which is a modified version of BPSO algorithm, and SA-PSO [128], a modified version of PSO algorithm, for allocation of the offloadable tasks to the backend edge servers.

Furthermore, all these results can be compared to study the outcomes of utilizing these various meta-heuristic techniques for task allocation, as such a comparison is a crucial step in finding the best approach for a specific scenario. This exploration is necessary as it will assist in determining which among these techniques is delivering the best results for various parameters like:

- efficiency, i.e., minimizing completion time or maximizing resource utilization
- quality of solution, i.e., to have an idea of how close the achieved allocation is to the optimal solution
- scalability, i.e., how the technique handles an increasing number of tasks or resources

This comparative analysis would allow researchers to have a holistic view of the problem as it is known that in optimization problems, there is no single best technique, but the best possible decision always depends on the problem's characteristics, parameters, and desired outcomes. At the same time, hybridization, i.e., merging elements from various meta-heuristic algorithm can make way for a more robust algorithm.

Overall, this research direction would be useful, as by systematically equating and making alterations in meta-heuristic techniques, the researchers will be able to enhance more effective and efficient results for task allocation problems over various domains.

Bibliography

[1] Raghu K Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: current state and future challenges. *IEEE communications Magazine*, 49(11):32–39, 2011.

[2] Bin Guo, Zhu Wang, Zhiwen Yu, Yu Wang, Neil Y Yen, Runhe Huang, and Xingshe Zhou. Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. *ACM computing surveys (CSUR)*, 48(1):1–31, 2015.

[3] Arpita Ray, Sakil Mallick, Sukanta Mondal, Soumik Paul, Chandreyee Chowdhury, and Sarbani Roy. A framework for mobile crowd sensing and computing based systems. In *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6. IEEE, 2018.

[4] Arpita Ray, Chandreyee Chowdhury, and Sarbani Roy. Strategic decision for crowd-sensing: An approach based on markov decision process. In *2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6. IEEE, 2017.

[5] Arpita Ray, Chandreyee Chowdhury, Sakil Mallick, Sukanta Mondal, Soumik Paul, and Sarbani Roy. Designing energy efficient strategies using markov decision process for crowd-sensing applications. *Mobile Networks and Applications*, 25:932–942, 2020.

[6] Xing Chen, Shihong Chen, Xuee Zeng, Xianghan Zheng, Ying Zhang, and

Chunming Rong. Framework for context-aware computation offloading in mobile cloud computing. *Journal of Cloud Computing*, 6:1–17, 2017.

[7] Venus Haghighi and Naghmeh S Moayedian. An offloading strategy in mobile cloud computing considering energy and delay constraints. *IEEE Access*, 6:11849–11861, 2018.

[8] Xiaomin Jin, Yuanan Liu, Wenhao Fan, Fan Wu, and Bihua Tang. Multisite computation offloading in dynamic mobile cloud environments. *Science China Information Sciences*, 60:1–3, 2017.

[9] Qiushi Wang and Katinka Wolter. Detection and analysis of performance deterioration in mobile offloading system. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, pages 420–425. IEEE, 2014.

[10] Tianhui Meng, Qiushi Wang, and Katinka Wolter. Model-based quantitative security analysis of mobile offloading systems under timing attacks. In *Analytical and Stochastic Modelling Techniques and Applications: 22nd International Conference, ASMTA 2015, Albena, Bulgaria, May 26-29, 2015. Proceedings 22*, pages 143–157. Springer, 2015.

[11] Chandreyee Chowdhury, Sarbani Roy, Arpita Ray, and Sumanta Kumar Deb. A fault-tolerant approach to alleviate failures in offloading systems. *Wireless Personal Communications*, 110(2):1033–1055, 2020.

[12] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE access*, 8:85714–85728, 2020.

[13] Jinwei Liu, Haiying Shen, Husnu S Narman, Wingyan Chung, and Zongfang Lin. A survey of mobile crowdsensing techniques: A critical component for the internet of things. *ACM Transactions on Cyber-Physical Systems*, 2(3):1–26, 2018.

[14] Nicholas D Lane, Yohan Chon, Lin Zhou, Yongzhe Zhang, Fan Li, Dongwon Kim, Guanzhong Ding, Feng Zhao, and Hojung Cha. Piggyback crowdsensing (pcs) energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14, 2013.

[15] Bin Guo, Zhiwen Yu, Xingshe Zhou, and Daqing Zhang. From participatory sensing to mobile crowd sensing. In *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*, pages 593–598. IEEE, 2014.

[16] Stefan Pietschmann, Annett Mitschick, Ronny Winkler, and Klaus Meißner. Croco: Ontology-based, cross-application context management. In *2008 Third International Workshop on Semantic Media Adaptation and Personalization*, pages 88–93. IEEE, 2008.

[17] Rabindra K Barik, Sudhansu Shekhar Patra, Rasmita Patro, Sachi Nandan Mohanty, and Abdulsattar Abdullah Hamad. Geobd2: Geospatial big data deduplication scheme in fog assisted cloud computing environment. In *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 35–41. IEEE, 2021.

[18] Pasquale Puzio, Refik Molva, Melek Önen, and Sergio Loureiro. Block-level deduplication with encrypted data. *Open Journal of Cloud Computing (OJCC)*, 1(1):10–18, 2014.

[19] Silambarasan Elkana Ebinazer, Nickolas Savarimuthu, and S Mary Saira Bhanu. Eskea: enhanced symmetric key encryption algorithm based secure data storage in cloud networks with data deduplication. *Wireless Personal Communications*, 117:3309–3325, 2021.

[20] Youngchul Kim, Cheiyol Kim, Sangmin Lee, and Youngkyun Kim. Design and implementation of inline data deduplication in cluster file system. *KIISE Transactions on Computing Practices*, 22(8):369–374, 2016.

[21] Avani Wildani, Ethan L Miller, and Ohad Rodeh. Hands: A heuristically arranged non-backup in-line deduplication system. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 446–457. IEEE, 2013.

[22] Kiran Srinivasan, Timothy Bisson, Garth R Goodson, and Kaladhar Voruganti. idedup: latency-aware, inline data deduplication for primary storage. In *Fast*, volume 12, pages 1–14, 2012.

[23] Tianming Yang, Hong Jiang, Dan Feng, Zhongying Niu, Ke Zhou, and Yaping Wan. Debar: A scalable high-performance de-duplication storage system for backup and archiving. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12. IEEE, 2010.

[24] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (ToS)*, 7(4):1–20, 2012.

[25] Atish Kathpal, Matthew John, and Gaurav Makkar. Distributed duplicate detection in post-process data de-duplication. In *HiPC*, 2011.

[26] Jurairat Phuttharak and Seng W Loke. A review of mobile crowdsourcing architectures and challenges: Toward crowd-empowered internet-of-things. *Ieee access*, 7:304–324, 2018.

[27] Yingjie Wang, Zhipeng Cai, Zhi-Hui Zhan, Yue-Jiao Gong, and Xiangrong Tong. An optimization and auction-based incentive mechanism to maximize social welfare for mobile crowdsourcing. *IEEE Transactions on Computational Social Systems*, 6(3):414–429, 2019.

[28] Zhibo Wang, Yuting Huang, Xinkai Wang, Ju Ren, Qian Wang, and Libing Wu. Socialrecruiter: Dynamic incentive mechanism for mobile crowdsourcing worker recruitment with social networks. *IEEE Transactions on Mobile Computing*, 20(5):2055–2066, 2020.

[29] Zhiyan Chen, Claudio Fiandrino, and Burak Kantarci. On blockchain integration into mobile crowdsensing via smart embedded devices: A comprehensive survey. *Journal of Systems Architecture*, 115:102011, 2021.

[30] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):1–55, 2014.

[31] Dennis Mohan Punjabi, Li-Ping Tung, and Bao-Shuh Paul Lin. Crowdsmile: a crowdsourcing-based social and mobile integrated system for learning by exploration. In *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, pages 521–526. IEEE, 2013.

[32] Dirk Eilander, Patricia Trambauer, Jurjen Wagemaker, and Arnejan Van Loenen. Harvesting social media for generation of near real-time flood maps. *Procedia Engineering*, 154:176–183, 2016.

[33] Rajib Kumar Rana, Chun Tung Chou, Salil S Kanhere, Nirupama Bulusu, and Wen Hu. Ear-phone: an end-to-end participatory urban noise mapping system. In *Proceedings of the 9th ACM/IEEE international conference on information processing in sensor networks*, pages 105–116, 2010.

[34] Minori Matsuyama, Ryuichi Nisimura, Hideki Kawahara, Junnosuke Yamada, and Toshio Irino. Development of a mobile application for crowdsourcing the data collection of environmental sounds. In *Human Interface and the Management of Information. Information and Knowledge Design and Evaluation:*

16th International Conference, HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings, Part I 16, pages 514–524. Springer, 2014.

[35] Jing Zhang, Victor S Sheng, Tao Li, and Xindong Wu. Improving crowd-sourced label quality using noise correction. *IEEE transactions on neural networks and learning systems*, 29(5):1675–1688, 2017.

[36] Ian McCallum, Linda See, Tobias Sturn, Carl Salk, Christoph Perger, Martina Duerauer, Mathias Karner, Inian Moorthy, Dahlia Domian, Dmitry Schepaschenko, et al. Engaging citizens in environmental monitoring via gaming. *International Journal of Spatial Data Infrastructures Research*, 13:15–23, 2018.

[37] Alessandro Grazioli, Marco Picone, Francesco Zanichelli, and Michele Amoretti. Collaborative mobile application and advanced services for smart parking. In *2013 IEEE 14th International Conference on Mobile Data Management*, volume 2, pages 39–44. IEEE, 2013.

[38] Daehan Kwak, Daeyoung Kim, Ruilin Liu, Liviu Iftode, and Badri Nath. Tweeting traffic image reports on the road. In *6th International Conference on Mobile Computing, Applications and Services*, pages 40–48. IEEE, 2014.

[39] Yao-Chung Fan, Cheng Teng Iam, Gia Hao Syu, and Wei Hong Lee. Teleeye: Enabling real-time geospatial query answering with mobile crowd. In *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, pages 323–324. IEEE, 2013.

[40] Darren Boss, Trisalyn Nelson, Meghan Winters, and Colin J Ferster. Using crowdsourced data to monitor change in spatial patterns of bicycle ridership. *Journal of Transport & Health*, 9:226–233, 2018.

[41] Ruo-Qian Wang, Huina Mao, Yuan Wang, Chris Rae, and Wesley Shaw.

Hyper-resolution monitoring of urban flooding with social media and crowdsourcing data. *Computers & Geosciences*, 111:139–147, 2018.

[42] Jack Reilly, Shideh Dashti, Mari Ervasti, Jonathan D Bray, Steven D Glaser, and Alexandre M Bayen. Mobile phones as seismologic sensors: Automating data extraction for the ishake system. *IEEE Transactions on Automation Science and Engineering*, 10(2):242–251, 2013.

[43] Kevin Gimpel, Nathan Schneider, Brendan O'connor, Dipanjan Das, Daniel P Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the 49th annual meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 42–47, 2011.

[44] Won-Jae Yi, Weidi Jia, and Jafar Saniie. Mobile sensor data collector using android smartphone. In *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 956–959. IEEE, 2012.

[45] Rüdiger Pryss, Manfred Reichert, Berthold Langguth, and Winfried Schlee. Mobile crowd sensing services for tinnitus assessment, therapy, and research. In *2015 IEEE international conference on mobile services*, pages 352–359. IEEE, 2015.

[46] Jaime Ballesteros, Bogdan Carbunar, Mahmudur Rahman, Naphtali Rishe, and SS Iyengar. Towards safe cities: A mobile and social networking approach. *IEEE Transactions on Parallel and Distributed Systems*, 25(9):2451–2462, 2013.

[47] Margaret Hamilton, Flora Salim, Eva Cheng, and Sue Lynn Choy. Transafe: A crowdsourced mobile platform for crime and safety perception management. *ACM Sigcas Computers and Society*, 41(2):32–37, 2011.

[48] Derek G Murray, Eiko Yoneki, Jon Crowcroft, and Steven Hand. The case for crowd computing. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, pages 39–44, 2010.

[49] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Honeybee: A programming framework for mobile crowd computing. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services: 9th International Conference, MobiQuitous 2012, Beijing, China, December 12-14, 2012. Revised Selected Papers 9*, pages 224–236. Springer, 2013.

[50] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future generation computer systems*, 29(1):84–106, 2013.

[51] Michael J Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72, 2011.

[52] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 77–90, 2010.

[53] Giuseppe Cardone, Antonio Corradi, Luca Foschini, and Raffaele Ianniello. Participact: A large-scale crowdsensing platform. *IEEE Transactions on Emerging Topics in Computing*, 4(1):21–32, 2015.

[54] Xiping Hu, Terry HS Chu, Henry CB Chan, and Victor CM Leung. Vita: A crowdsensing-oriented mobile cyber-physical system. *IEEE Transactions on Emerging Topics in Computing*, 1(1):148–165, 2013.

[55] Charith Perera, Dumidu S. Talagala, Chi Harold Liu, and Julio C. Estrella. Energy efficient location and activity-aware on-demand mobile distributed sensing platform for sensing as a service in iot clouds. *IEEE Transactions on Computational Social Systems*, 2(4):171–181, 2015.

[56] Changkun Jiang, Lin Gao, Lingjie Duan, and Jianwei Huang. Economics of peer-to-peer mobile crowdsensing. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2015.

[57] Joaquin Guillen, Javier Miranda, Javier Berrocal, Jose Garcia-Alonso, Juan Manuel Murillo, and Carlos Canal. People as a service: a mobile-centric model for providing collective sociological profiles. *IEEE software*, 31(2):48–53, 2013.

[58] Mohamed Abdelaal, Mohammad Qaid, Frank Dürr, and Kurt Rothermel. isense: Energy-aware crowd-sensing framework. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, pages 1–9. IEEE, 2017.

[59] Alireza Hassani, Pari Delir Haghighi, Prem Prakash Jayaraman, and Arkady Zaslavsky. A context aware framework for mobile crowd-sensing. In *Modeling and Using Context: 10th International and Interdisciplinary Conference, CONTEXT 2017, Paris, France, June 20-23, 2017, Proceedings 10*, pages 557–568. Springer, 2017.

[60] Weiwei Wu, Jianping Wang, Minming Li, Kai Liu, Feng Shan, and Junzhou Luo. Energy-efficient transmission with data sharing in participatory sensing systems. *IEEE Journal on Selected Areas in Communications*, 34(12):4048–4062, 2016.

[61] Jing Wang, Jian Tang, Guoliang Xue, and Dejun Yang. Towards energy-

efficient task scheduling on smartphones in mobile crowd sensing systems. *Computer Networks*, 115:100–109, 2017.

[62] Ahmed Abdel Moamen and Nadeem Jamali. Share sens: An approach to optimizing energy consumption of continuous mobile sensing workloads. In *2015 IEEE International Conference on Mobile Services*, pages 89–96. IEEE, 2015.

[63] Andrea Capponi, Claudio Fiandrino, Dzmitry Kliazovich, Pascal Bouvry, and Stefano 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.

[64] Chi Harold Liu, Bo Zhang, Xin Su, Jian Ma, Wendong Wang, and Kin K Leung. Energy-aware participant selection for smartphone-enabled mobile crowd sensing. *IEEE Systems Journal*, 11(3):1435–1446, 2015.

[65] Leye Wang, Daqing Zhang, Zhixian Yan, Haoyi Xiong, and Bing Xie. effsense: A novel mobile crowd-sensing framework for energy-efficient and cost-effective data uploading. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(12):1549–1563, 2015.

[66] Chenren Xu, Vijay Srinivasan, Jun Yang, Yoshiya Hirase, Emmanuel Munguia Tapia, and Yanyong Zhang. Boe: Context-aware global power management for mobile devices balancing battery outage and user experience. In *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 300–308. IEEE, 2014.

[67] Mengxi Zhang, Yanjie Li, and Haoyao Chen. A semi-markov decision process based dynamic power management for mobile devices. In *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 249–254. IEEE, 2016.

[68] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.

[69] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62, 2010.

[70] Huber Flores, Pan Hui, Sasu Tarkoma, Yong Li, Satish Srirama, and Rajkumar Buyya. Mobile code offloading: from concept to practice and beyond. *IEEE Communications Magazine*, 53(3):80–88, 2015.

[71] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf, 2009.

[72] Huaming Wu, Qiushi Wang, and Katinka Wolter. Tradeoff between performance improvement and energy saving in mobile cloud offloading systems. In *2013 IEEE international conference on communications workshops (ICC)*, pages 728–732. IEEE, 2013.

[73] Nikita Bhagat and Shashi Bhushan. Energy efficient offloading for mobile cloud computing: A review. *International Journal of Advanced Engineering Research and Applications.*, 2(2):65–71, 2016.

[74] Michael L Powell and Barton P Miller. Process migration in demos/mp. *ACM SIGOPS Operating Systems Review*, 17(5):110–119, 1983.

[75] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey

of computation offloading for mobile systems. *Mobile networks and Applications*, 18:129–140, 2013.

[76] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.

[77] Shuai Yu, Rami Langar, Wenzhong Li, and Xu Chen. Coalition-based energy efficient offloading strategy for immersive collaborative applications in femto-cloud. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.

[78] Shuai Yu, Rami Langar, and Xin Wang. A d2d-multicast based computation offloading framework for mobile edge computing. In *IEEE GLOBECOM*, 2016.

[79] Shumao Ou, Yumin Wu, Kun Yang, and Bosheng Zhou. Performance analysis of fault-tolerant offloading systems for pervasive services in mobile wireless environments. In *2008 IEEE international conference on communications*, pages 1856–1860. IEEE, 2008.

[80] Sayanti Mondal, Chandreyee Chowdhury, Sarbani Roy, Sumanta K Deb, and Sarmistha Neogy. Crash failure immune offloading framework. In *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6. IEEE, 2016.

[81] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[82] Manshu Goyal and Poonam Saini. A fault-tolerant energy-efficient computational offloading approach with minimal energy and response time in mobile cloud computing. In *2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pages 44–49. IEEE, 2016.

[83] Sura Khalil Abd, Syed Abdul Rahman Al-Haddad, Fazirulhisyam Hashim, Azizol BHJ Abdullah, and Salman Yussof. Energy-aware fault tolerant task offloading of mobile cloud computing. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 161–164. IEEE, 2017.

[84] Mark S Gordon, D Anoushe Jamshidi, Scott Mahlke, Z Morley Mao, and Xu Chen. {COMET}: Code offload by migrating execution transparently. In *10th USENIX symposium on operating systems design and implementation (OSDI 12)*, pages 93–106, 2012.

[85] Yennun Huang, Chandra Kintala, Nick Kolettis, and N Dudley Fulton. Software rejuvenation: Analysis, module and applications. In *Twenty-fifth international symposium on fault-tolerant computing. Digest of papers*, pages 381–390. IEEE, 1995.

[86] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. Software aging and rejuvenation: Where we are and where we are going. In *2011 IEEE Third International Workshop on Software Aging and Rejuvenation*, pages 1–6. IEEE, 2011.

[87] Aad PA Van Moorsel and Katinka Wolter. Analysis of restart mechanisms in software systems. *IEEE Transactions on Software Engineering*, 32(8):547–558, 2006.

[88] Shumao Ou, Kun Yang, and Liang Hu. Cross: A combined routing and surrogate selection algorithm for pervasive service offloading in mobile ad hoc environments. In *IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference*, pages 720–725. IEEE, 2007.

[89] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the 2001*

international conference on Compilers, architecture, and synthesis for embedded systems, pages 238–246, 2001.

[90] Chester Rebeiro, Debdeep Mukhopadhyay, and Sarani Bhattacharya. *Timing channels in cryptography: a micro-architectural perspective*. Springer, 2014.

[91] Jibang Liu and Yung-Hsiang Lu. Energy savings in privacy-preserving computation offloading with protection by homomorphic encryption. In *Proceedings of the 2010 international conference on Power aware computing and systems, HotPower*, volume 10, pages 1–7, 2010.

[92] Jibang Liu, Karthik Kumar, and Yung-Hsiang Lu. Tradeoff between energy savings and privacy protection in computation offloading. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, pages 213–218, 2010.

[93] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. Fuzzy workload orchestration for edge computing. *IEEE Transactions on Network and Service Management*, 16(2):769–782, 2019.

[94] Md Delowar Hossain, Tangina Sultana, VanDung Nguyen, Waqas ur Rahman, Tri DT Nguyen, Luan NT Huynh, and Eui-Nam Huh. Fuzzy based collaborative task offloading scheme in the densely deployed small-cell networks with multi-access edge computing. *Applied Sciences*, 10(9):3115, 2020.

[95] Xiaofeng Cao, Guoming Tang, Deke Guo, Yan Li, and Weiming Zhang. Edge federation: Towards an integrated service provisioning model. *IEEE/ACM Transactions on Networking*, 28(3):1116–1129, 2020.

[96] Ermioni Qafzezi, Kevin Bylykbashi, Phudit Ampririt, Makoto Ikeda, Keita Matsuo, and Leonard Barolli. An intelligent approach for cloud-fog-edge com-

puting sdn-vanets based on fuzzy logic: effect of different parameters on coordination and management of resources. *Sensors*, 22(3):878, 2022.

[97] Yujiong Liu, Shangguang Wang, Qinglin Zhao, Shiyu Du, Ao Zhou, Xiao Ma, and Fangchun Yang. Dependency-aware task scheduling in vehicular edge computing. *IEEE Internet of Things Journal*, 7(6):4961–4971, 2020.

[98] Jungmin Son, Amir Vahid Dastjerdi, Rodrigo N Calheiros, and Rajkumar Buyya. Sla-aware and energy-efficient dynamic overbooking in sdn-based cloud data centers. *IEEE Transactions on Sustainable Computing*, 2(2):76–89, 2017.

[99] Sanghyeok Kim, Sungyoung Park, Youngjae Kim, Siri Kim, and Kwonyong Lee. Vnf-eq: dynamic placement of virtual network functions for energy efficiency and qos guarantee in nfv. *Cluster Computing*, 20:2107–2117, 2017.

[100] Leonard Nonde, Taisir EH El-Gorashi, and Jaafar MH Elmirghani. Energy efficient virtual network embedding for cloud networks. *Journal of Lightwave Technology*, 33(9):1828–1849, 2014.

[101] Onur Ascigil, Truong Khoa Phan, Argyrios G Tasiopoulos, Vasilis Sourlas, Ioannis Psaras, and George Pavlou. On uncoordinated service placement in edge-clouds. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 41–48. IEEE, 2017.

[102] Jungmin Son and Rajkumar Buyya. Latency-aware virtualized network function provisioning for distributed edge clouds. *Journal of Systems and Software*, 152:24–31, 2019.

[103] Jun Cheng and Dejun Guan. Research on task-offloading decision mechanism in mobile edge computing-based internet of vehicle. *EURASIP Journal on Wireless Communications and Networking*, 2021(1):1–14, 2021.

[104] Qian You and Bing Tang. Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things. *Journal of Cloud Computing*, 10:1–11, 2021.

[105] Md Sajjad Hossain, Cosmas Ifeanyi Nwakanma, Jae Min Lee, and Dong-Seong Kim. Edge computational task offloading scheme using reinforcement learning for iiot scenario. *ICT Express*, 6(4):291–299, 2020.

[106] Wen-Hsing Kuo and Yung-Cheng Wang. An energy-saving edge computing and transmission scheme for iot mobile devices. In *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, pages 1–2. IEEE, 2019.

[107] Jinfang Sheng, Jie Hu, Xiaoyu Teng, Bin Wang, and Xiaoxia Pan. Computation offloading strategy in mobile edge computing. *Information*, 10(6):191, 2019.

[108] Jinming Wen, Chao Ren, and Arun Kumar Sangaiah. Energy-efficient device-to-device edge computing network: An approach offloading both traffic and computation. *IEEE communications magazine*, 56(9):96–102, 2018.

[109] Rasim M Alguliyev, Yadigar N Imamverdiyev, and Fargana J Abdullayeva. Pso-based load balancing method in cloud computing. *Automatic Control and Computer Sciences*, 53:45–55, 2019.

[110] Chendong Liu, Yilin Zhang, and Huanyu Zhou. A comprehensive study of bluetooth low energy. In *Journal of Physics: Conference Series*, volume 2093, page 012021. IOP Publishing, 2021.

[111] Xinglin Zhang, Zheng Yang, Wei Sun, Yunhao Liu, Shaohua Tang, Kai Xing, and Xufei Mao. Incentives for mobile crowd sensing: A survey. *IEEE Communications Surveys & Tutorials*, 18(1):54–67, 2015.

[112] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[113] Elena Pashenkova, Irina Rish, and Rina Dechter. Value iteration and policy iteration algorithms for markov decision problem. In *AAAI'96: Workshop on Structural Issues in Planning and Temporal Reasoning*. Citeseer, 1996.

[114] Alireza Seyedi and Biplab Sikdar. Energy efficient transmission strategies for body sensor networks with energy harvesting. *IEEE Transactions on Communications*, 58(7):2116–2126, 2010.

[115] Priya Roy, Chandreyee Chowdhury, Dip Ghosh, and Sanghamitra Bandyopadhyay. Juindoorloc: A ubiquitous framework for smartphone-based indoor localization subject to context and device heterogeneity. *Wireless Personal Communications*, 106:739–762, 2019.

[116] Robert Sheahan, Lester Lipsky, Pierre M Fiorini, and Søren Asmussen. On the completion time distribution for tasks that must restart from the beginning if a failure occurs. *ACM SIGMETRICS Performance Evaluation Review*, 34(3):24–26, 2006.

[117] PowerTutor. `http://ziyang.eecs.umich.edu/projects/powertutor/`. [Accessed 10-11-2023].

[118] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[119] Daniele Santoro, Daniel Zozin, Daniele Pizzolli, Francesco De Pellegrini, and Silvio Cretti. Foggy: A platform for workload orchestration in a fog computing environment. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 231–234. IEEE, 2017.

[120] Huber Flores, Xiang Su, Vassilis Kostakos, Aaron Yi Ding, Petteri Nurmi, Sasu Tarkoma, Pan Hui, and Yong Li. Large-scale offloading in the internet of things. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 479–484. IEEE, 2017.

[121] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[122] Emile Aarts and Jan Korst. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing.* John Wiley & Sons, Inc. ISBN: 978-0-471-92146-2, 1989.

[123] Gamal Attiya and Yskandar Hamam. Task allocation for maximizing reliability of distributed systems: A simulated annealing approach. *Journal of parallel and Distributed Computing*, 66(10):1259–1266, 2006.

[124] Shivani Deshmukh and Lakshay Grover. Implementation & analysis of pureedgesim. *International Journal of Engineering Applied Sciences and Technology*, 5(7):129–133, 2020.

[125] Rob A Rutenbar. Simulated annealing algorithms: An overview. *IEEE Circuits and Devices magazine*, 5(1):19–26, 1989.

[126] Yaghout Nourani and Bjarne Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373, 1998.

[127] Hossein Nezamabadi-pour, Majid Rostami-Shahrbabaki, and Malihe Maghfoori-Farsangi. Binary particle swarm optimization: challenges and new solutions. *CSI J Comput Sci Eng*, 6(1):21–32, 2008.

[128] S Sudibyo, MN Murat, and Norashid Aziz. Simulated annealing-particle swarm optimization (sa-pso): Particle distribution study and application in neural wiener-based nmpc. In *2015 10th Asian Control Conference (ASCC)*, pages 1–6. IEEE, 2015.