DYNAMIC AND ADAPTIVE EXECUTION MODELS FOR DATA STREAM MINING APPLICATIONS IN MOBILE EDGE CLOUD COMPUTING SYSTEMS

MUHAMMAD HABIB UR REHMAN

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

DYNAMIC AND ADAPTIVE EXECUTION MODELS FOR DATA STREAM MINING APPLICATIONS IN MOBILE EDGE CLOUD COMPUTING SYSTEMS

MUHAMMAD HABIB UR REHMAN

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Muhammad Habib ur Rehman

Registration/Matrix No.: WHA130014

Name of Degree: Doctor of Philosophy

Title of Thesis: Dynamic and Adaptive Execution Models for Data Stream Mining Applications in Mobile Edge Cloud Computing Systems

Field of Study: Distributed Computing

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date

Subscribed and solemnly declared before,

Witness's Signature Date

Name:
Designation:

ABSTRACT

Mobile edge cloud computing (MECC) systems extend computational, networking, and storage capabilities of centralized cloud computing systems through edge servers at one-hop wireless distances from mobile devices. Mobile data stream mining (MDSM) applications in MECC systems involve massive heterogeneity at application and platform levels. At application level, the program components need to handle continuously streaming data in order to perform knowledge discovery operations. At platform level, the MDSM applications need to seamlessly switch the execution processes among mobile devices, edge servers, and cloud computing servers. However, the execution of MDSM applications in MECC systems becomes hard due to multiple factors. The critical factors of complexity at application level include data size and data rate of continuously streaming data, the selection of data fusion and data preprocessing methods, the choice of learning models, learning rates and learning modes, and the adoption of data mining algorithms. Alternately, the platform level complexity increases due to mobility and limited availability of computational and battery power resources in mobile devices, high coupling between application components, and dependency over Internet connections. Considering the complexity factors, existing literature proposes static execution models for MDSM applications. The execution models are based on either standalone mobile devices, mobile-to-mobile, mobile-to-edge, or mobile-to-cloud communication models. This thesis presents the novel architecture which utilizes far-edge mobile devices as a primary execution platform for MDSM applications. At the secondary level, the architecture executes MDSM applications by enabling direct communication among nearer mobile devices through local Wi-Fi routers without connecting to the Internet. At tertiary level, the architecture enables far-edge to cloud communication in case of unavailability of onboard computational and battery power resources and in the absence of any other

mobile devices in the locality. This thesis also presents the dynamic and adaptive execution models in order to handle the complexity at application and platform levels. The dynamic execution model facilitates the data-intensive MDSM applications having low computational complexity. However, the adaptive execution model facilitates in seamless execution of MDSM applications having low data-intensity but high computational complexities. Multiple evaluation methods were used in order to verify and validate the performance of proposed architecture and execution models. The validation and verification of the proposed architecture were performed using High-Level Petri Nets (HLPN) and Z3 Solver. The simulation results revealed that all states in the HLPN model were reachable and the overall design presented a workable solution. However, proposed architecture faced the state explosion problem wherein conventional static execution models fail because the system may enter in multiple states of execution from a single state. The proposed dynamic and adaptive execution models help address the issue of the state explosion problem. To this end, the proposed execution models were tested with multiple MDSM applications mapping to a real-world use-case for activity detection using MECC systems. The experimental evaluation was made in terms of battery power consumption, memory utilization, makespan, accuracy, and the amount of data reduced in mobile devices. The comparison showed that proposed dynamic and adaptive execution models outperformed the static execution models in multiple aspects.

ABSTRAK

Sistem MECC menambah baik sistem pengkomputeran, rangkaian, dan keupayaan penyimpanan sistem pengkomputeran awan berpusat yang melalui pelayan sempadan pada jarak satu-hop tanpa wayar dari peranti mudah alih. Aplikasi MDSM di dalam sistem MECC melibatkan kepelbagaian yang banyak di peringkat aplikasi dan platform. Pada peringkat aplikasi, komponen program perlu menguruskan aliran data secara berterusan dalam melaksanakan operasi penemuan pengetahuan. Pada peringkat platform pula, aplikasi MDSM perlulah lancar dalam menukar proses pelaksanaan di antara peranti mudah alih, pelayan sempadan, dan pelayan pengkomputeran awan. Walaubagaimanapun, pelaksanaan aplikasi MDSM di dalam sistem MECC menjadi sukar disebabkan beberapa faktor. Faktor kritikal yang rumit di tahap aplikasi merangkumi saiz data dan kadar aliran data yang berterusan, pemilihan gabungan data dan kaedah-kaedah pemprosesan awal data, pilihan model pembelajaran, kadar dan mod pembelajaran, dan penggunaan algoritma perlombongan data. Kemudian, kerumitan di tahap platform meningkat kerana mobiliti begitu juga penyediaan sumber pengkomputeran dan kuasa bateri dalam peranti mudah alih yang terhad, gandingan tinggi antara komponen aplikasi, dan kebergantungan kepada sambungan Internet. Berikutan itu, kajian yang sedia ada mencadangkan model pelaksanaan statik untuk aplikasi MDSM. Model-model pelaksanaan adalah sama ada berdasarkan peranti mudah alih 'standalone mobile devices', mudah alih ke mudah alih 'mobile-to-mobile', mudah alih ke sempadan 'mobile-to-edge', atau model komunikasi mudah alih ke awan 'mobile-to-cloud'. Tesis ini membentangkan arkitektur baru yang menggunakan peranti mudah alih 'far-edge' sebagai platform pelaksanaan utama untuk aplikasi MDSM. Di peringkat sekunder, pelaksanaan aplikasi MDSM adalah dengan membenarkan komunikasi secara langsung antara peranti mudah alih yang terdekat melalui router Wi-Fi tanpa penyambungan ke Internet. Pada peringkat tertier pula, ia

membenarkan ‘far-edge’ kepada komunikasi awan sekiranya kekurangan sumber pengkomputeran dan sumber kuasa dan dalam ketiadaan mana-mana peranti mudah alih berdekatan. Di samping itu, tesis ini membentangkan model pelaksanaan yang dinamik dan adaptif untuk mengendalikan kerumitan di peringkat aplikasi dan platform. Model pelaksanaan dinamik ini memudahkan aplikasi data intensif MDSM mempunyai kerumitan pengkomputeran yang minima. Walau bagaimanapun, model pelaksanaan adaptif di dalam aplikasi MDSM ini mempunyai data berintensiti rendah tetapi pengiraan berintensiti tinggi. Pelbagai kaedah penilaian telah digunakan untuk mengukur dan mengesahkan prestasi dan pelaksanaan model yang dicadangkan. Verifikasi dan pengesahan cadangan kajian dilakukan menggunakan High-Level Petri Nets (HLPN) dan Z3 Solver. Hasil keputusan simulasi menunjukkan semua keadaan di dalam model HLPN adalah tercapai dan reka bentuk keseluruhan menunjukkan solusi yang boleh diguna pakai. Walau bagaimanapun, model yang dicadangkan menghadapi masalah dimana model pelaksanaan statik konvensional gagal kerana sistem itu boleh memasuki dalam pelbagai keadaan pelaksanaan dari keadaan yang tunggal. Dengan model pelaksanaan dinamik dan adaptif yang dicadangkan ini dapat membantu menangani isu masalah seperti itu. Untuk tujuan ini, model pelaksanaan yang dicadangkan telah diuji dengan pelbagai aplikasi MDSM disesuaikan dengan penggunaan kes dunia sebenar untuk mengesan aktiviti menggunakan sistem MECC. Penilaian eksperimen telah dibuat dari segi penggunaan kuasa bateri, penggunaan memori, ‘makespan’, dan jumlah data yang dikurangkan dalam peranti mudah alih. Perbandingan menunjukkan bahawa cadangan model pelaksanaan dinamik dan adaptif mengatasi model pelaksanaan statik dari pelbagai aspek.

ACKNOWLEDGEMENTS

Allah, the most beneficient, the most compassionate and merciful whose benevolence and blessings enabled me to accomplish this task.

I would like to express heartfelt gratitude to my supervisors Dr. Liew Chee Sun and Dr. Teh Ying Wah. Their vision, expertise, support, attention, and hard work guided me to the successful completion of this thesis. I am thankful to their consistent efforts and true desire to keep me on track.

I express my heartfelt gratitude to my parents (Hafiz Muhammad Ramzan and Naseem Akhtar), family, and friends for their love, prayers, moral support, encouragements, and sincere wishes for the completion of my work.

I am deeply indebted to Dr. Prem Prakash Jayaraman (Swinburne University of Technology, Australia), Dr. Saif ur Rehman Malik (COMSATS Institute of IT, Islamabad), Dr. Atta ur Rehman Khan (King Saud University, KSA), and Miss Aisha Batool (Iqra University, Islamabad) whose guidance, suggestions, and encouragements remained continual source of inspiration for me throughout the course of study.

I am grateful to University of Malaya (UM) for providing me a tremendous opportunity, guidance, and continuous motivational, financial, and technical support. I am thankful to CIIT for providing us a plateform to get this opportunity.

I am thankful to all the colleages at University of Malaya for their love, support, and giving me a quality time in Malaysia. I wish them happiness and success throughout their life.

Finally, I would like to dedicate this thesis to my beloved son "Muhammad Qasim Salar" to whom I always look as a source of motivation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER 1: INTRODUCTION

Personal data mining (also known as Personal Analytics and Quantified-Self) is a relatively new concept that is based on data mining techniques used for knowledge discovery from users' personal data to fulfill their personal needs (Ozzie et al., 2011). Personal data mining applications help in analyzing personal data and create wealth of opportunities to facilitate users in taking advantage from their personal knowledge patterns. Recently, a rapid growth can be observed in the development of personal data mining technologies and algorithms, as evidenced by quantified-self movement by Kevin Kelly and personal analytics by Stephen Wolfram (Choe, Lee, Lee, Pratt, & Kientz, 2014). At present, computational power and memory volume continues to be expanding, suggesting that personal data mining would become more feasible in the near future and attract more research attention (Gaber, Gama, Krishnaswamy, Gomes, & Stahl, 2014).

Cloud computing (CC) systems are defined as highly virtualized representation of a collection of interconnected computing and storage nodes in parallel and distributed systems (Buyya, Yeo, Venugopal, Broberg, & Brandic, 2009). The CC systems were traditionally designed as the utility computing model to meet the business needs of enterprises that required extra computational, networking, and storage services (Armbrust et al., 2010). However with the advent of mobile technologies, CC systems evolved into mobile cloud computing (MCC) systems whereby mobile devices harness CC resources to meet the resource limitations in terms of memory, CPU, and battery power (Fernando, Loke, & Rahayu, 2013). However, the multi-hop distances between mobile devices and computing resources in CC and MCC communication models increase the makespan (*i.e.* the time to process the data) in MCC applications (Satyanarayanan, Bahl, Caceres, & Davies, 2009). In addition, the transmission of continuously streaming data in CC and MCC systems increases the bandwidth utilization and data communication cost in MCC

communication model (A. R. Khan, Othman, Madani, & Khan, 2014) which further increases network traffic inside CC systems (Ahmed et al., 2015).

The edge cloud computing (ECC) systems extend the computational, networking and storage capabilities of CC systems to the edge of the Internet through edge servers (see Figure 1.1) (Satyanarayanan et al., 2009). Edge servers reside at one-hop wireless distances from mobile devices, which include smartphones, wearable devices, Internet of Things (IoTs) and wireless sensor networks, among many others (A. Ahmad & Ahmad, 2016). The provision of edge servers in close proximity of mobile devices reduces makespan in ECC applications (Satyanarayanan et al., 2009). In addition, mobile devices consume minimum battery power because the sensed data is immediately transferred to the edge servers for further processing (Fernando et al., 2013). Despite the reduced makespan and energy efficiency, the ECC communication model is Internet-dependent and always requires live Internet connection for data communication between edge servers and cloud data centers (Bonomi, Milito, Zhu, & Addepalli, 2012). The data communication between far-edge mobile devices and edge servers is performed using multiple communication protocols and interfaces such as Wi-Fi, Wi-Fi direct, Zigbee, Bluetooth low energy, 4G/5G, message queue telemetry transport (MQTT), and advanced message queuing protocol (AMQP), to name a few (Al-Fuqaha, Khreishah, Guizani, Rayes, & Mohammadi, 2015).

The vision of ECC is still unclear, but some supporting architectures are emerging. Such architectures introduce different topological settings as well as different notations (Ha & Satyanarayanan, 2015; Shaukat, Ahmed, Anwar, & Xia, 2015). In academia, edge servers are called cloudlets, which provide virtual cloud resources on the edge of a communication network (Satyanarayanan et al., 2009). In industry, Cisco defined ECC as Fog Computing, whereby gateway devices such as routers, switches and network access points work as edge servers (Bonomi et al., 2012). Similarly, Microsoft introduced

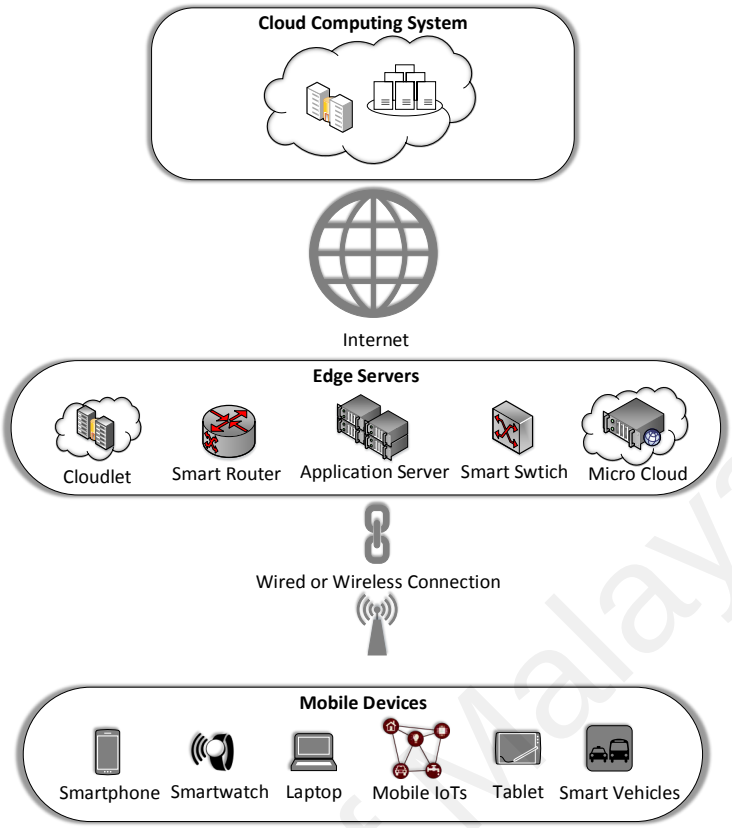


Figure 1.1: MECC Architecture.

micro-data centers as edge servers based on single servers, racks of servers, or content delivery networks with additional computational capabilities (Bahl, 2015). Intel envisions ECC whereby gateway devices are used as edge servers which filter and analyze the data streams before transferring in cloud data centers (Al-Fuqaha et al., 2015). The European Telecom Standards Institute deemed ECC as a standard architecture for the provision of remote cloud services in edge servers to enable communication through radio access networks over 3G/4G/5G data connections (Klas, 2015).

1.1 Problem Statement

Considering the traditional CC, MCC, and ECC systems, the execution of mobile data stream mining (MDSM) applications (as defined later in Section 2.1) is a challenging task due to continuously streaming data and the mobility of devices (Gaber, Gama, et al., 2014). The MDSM applications fail when the devices leave the network coverage

area (Satyanarayanan, 2015). In addition, the cloud resource manager needs to ensure the availability of virtual machines in close proximity of mobile devices in order to facilitate in seamless application execution (Bittencourt, Lopes, Petri, & Rana, 2015). The virtual machine migration introduces the resource management and high makespan overheads in MDSM applications (Satyanarayanan et al., 2009). Moreover the continuous data transfer in edge servers and cloud data centers increases the network traffic which results in high cost of data communication in terms of energy consumption and bandwidth utilization in CC systems. Using mobile devices as primary execution platform for MDSM applications can help in addressing these issues. Mobile devices contain bounded resources in terms of onboard memory and battery power. Existing studies present the adaptive execution strategies and light-weight data mining algorithms in order to achieve energy and memory efficiency in mobile devices (P. D. Haghighi et al., 2013). However, MDSM applications are enforced to compromise on quality of knowledge patterns (P. D. Haghighi et al., 2013). Considering the above mentioned issues and limitations in mobile devices, ECC, MCC, and CC systems, a new architecture is envisioned in this thesis (Rehman, Liew, Wah, Iqbal, & Jayaraman, 2016). The architecture utilizes mobile devices as primary platform for seamless execution of MDSM applications. In case of resource limitations the applications are executed at mobile edge servers at secondary level and at CC servers at tertiary level.

The MDSM applications need to handle continuously streaming data but onboard resource dynamics and mobility require efficient data management in order to ensure maximum data availability. Using a transient data management scheme where the data streams are temporarily stored in mobile devices helps address this issue. Considering the anatomy of MDSM application (as presented in Section 2.1), the computational complexities and resource utilizations of application components vary. Therefore sometimes the MDSM applications need to handle high data rate but the computational complexity

remains low. Such situations make mobile devices a favorite platform for application execution due to low latency and maximum data processing in mobile devices. Using a dynamic execution model, which monitors the onboard resources in mobile devices and resource utilization of MDSM applications, can help in maximum data processing in mobile environments. The dynamic model performs cost benefit analysis and offloads data streams to edge servers and CC servers, if necessary. Since the dynamic execution model forcefully executes few application components in far-edge mobile devices. Therefore, it needs to comprise on efficiency in terms of memory and battery power consumption when the computational complexities of application components increase. Using an adaptive execution model can help addressing this issue. The model can monitor and profile resource utilizations of multiple application components and run cost benefit analysis for data stream offloading from each component.

1.2 Mobile Edge Cloud Computing System

Mobile edge cloud computing (MECC) Systems extend the cloud services on the edge of the Internet through mobile and immobile edge servers that reside at one-hop wireless distances from mobile devices. The MECC systems enable distributed MDSM applications by extending CC services in edge servers as well as applications are partitioned at multiple levels (Ye, Ganti, Dimaghani, Grueneberg, & Calo, 2012). The MECC based MDSM applications span over far-edge mobile devices, edge servers, and traditional CC infrastructures (Ha et al., 2014).

Conventional immobile edge servers improve the battery lifetime of mobile devices and minimizes makespan in mobile applications. However, a persistent connectivity through Internet is required for mobile to edge and edge to cloud communication. Therefore the envisioned MECC architecture utilizes nearer mobile devices as edge servers and conventional CC systems for provision of cloud services (see Figure 1.2). The MECC

architecture enables mobile to edge communication using local communication channels such as Bluetooth, Bluetooth Low Energy, Wi-Fi, and Wi-Fi direct. Although the availability of mobile devices is an issue however the nearer mobile devices can be opportunistically used as mobile edge servers. The MECC architecture operates at three layers for: 1) *local execution*, whereby personal mobile devices are primarily used as application execution platforms, 2) *collaborative execution*, whereby the mobile edge servers are used as secondary level platforms for application execution, and 3) *remote execution*, whereby at tertiary level, the CC servers execute applications.

Opportunities: The MECC systems provide scalable computing infrastructure which can help in the deployment of highly distributed MDSM applications (Ye et al., 2012). Far-edge mobile devices in MECC systems perform single-site and multiple-site computation offloading (Simoens et al., 2013). In addition, the unlimited computational and storage support from traditional infrastructure based CC systems enable to deploy and dedi-

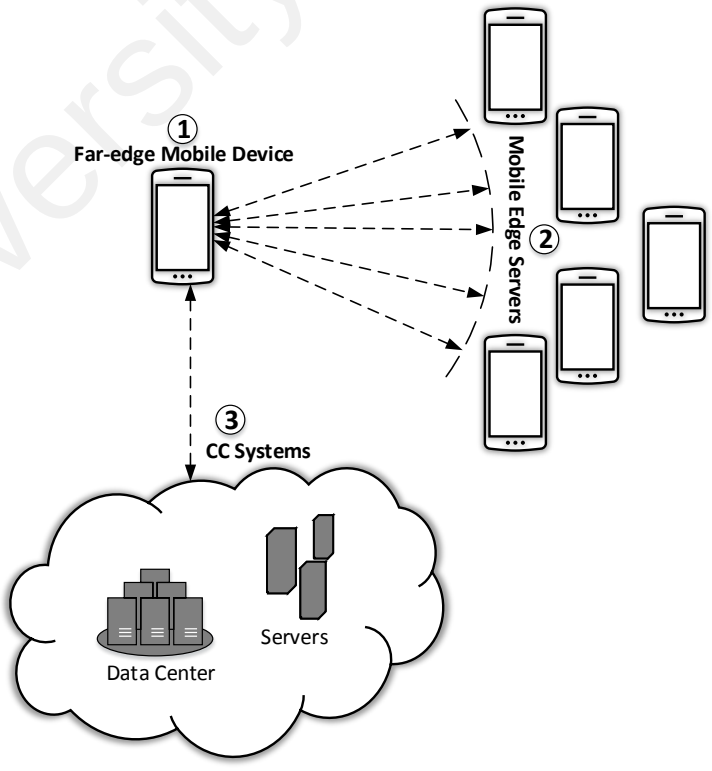


Figure 1.2: Reference MECC Architecture.

cate heterogeneous resources for edge servers (Ortiz, Huang, & Chakraborty, 2015). The edge servers can further utilize the acquired resources for seamless application execution. Edge servers also perform the resource-intensive computations to prolong battery life time and minimize makespan in MDSM applications (Drolia et al., 2013). Furthermore, the MDSM applications are geographically distributed to minimize the load-balancing efforts in an infrastructure based cloud (Luan, Gao, Li, Xiang, & Sun, 2015).

Challenges: The MDSM applications need to address several issues in MECC systems. The mobility and resource constraints in far-edge mobile devices enforce in location-aware predictive offloading of data streams and application components to minimize the makespan which emerges due to live virtual machines migrations and application state saving/resumption operations (Luan et al., 2015; Ha & Satyanarayanan, 2015). The high dependency of far-edge mobile devices on the availability of Internet connections may cause data loss as MDSM applications operate in online manner *i.e.* application components are executed as in-memory operations. Efficiently managing the raw and intermediate data streams help in improving overall knowledge quality in MDSM applications.

1.3 Objectives of Thesis and Research Methodology

This thesis has been articulated around four research objectives:

- **OB1:** To conduct experimental studies for feasibility and performance analysis of MDSM applications in far-edge mobile devices.
- **OB2:** To design, verify, validate, and develop a three-tier MECC architecture for MDSM applications.
- **OB3:** To design and develop an execution model for proposed architecture in order to dynamically execute MDSM applications in three-tier MECC architecture.
- **OB4:** To design and develop an execution model to enable MDSM applications for

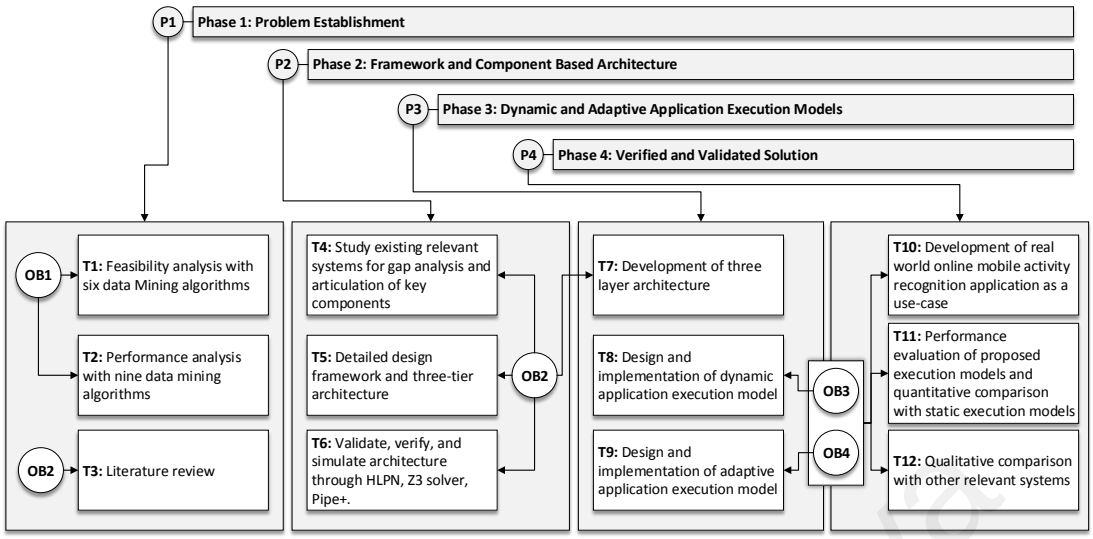


Figure 1.3: Research Methodology.

adaptive switching among far-edge mobile devices, mobile edge servers and CC servers.

To meet the four objectives (*i.e.* OB1, OB2, OB3, and OB4) in this thesis, the research was conducted in four phases (see Figure 1.3) whereby Task-1 to Task-3 (*i.e.* T1, T2, and T3) were performed at first phase to establish and formulate the research problem. At the second phase of this research, the tasks T4, T5, and T6 were performed to design the framework and three-tier component based architecture for distributed MDSM applications. At third phase, the tasks T7, T8, and T9 were performed to develop the proposed MECC architecture, and dynamic and adaptive execution models for MDSM applications. Finally, at forth phase, the tasks T10, T11, and T12 were performed to verify and validate the proposed solutions and perform quantitative and qualitative comparison with earlier static execution models for MDSM applications.

1.4 Thesis Contributions

The contributions of this thesis are:

1. A thorough literature review of MDSM applications and platforms have been performed. To this end, different topological settings of MDSM platforms have been

discussed. In addition, the execution models have been evaluated in multiple perspectives including model type, model behavior, and granularity of data processing. Moreover, a detailed review of data management and adaptation strategies, computation offloading methods, and data transfer and application partitioning schemes has been performed. Finally, a detailed gap analysis has been presented in order to find the research problem (Rehman, Liew, Wah, Shuja, & Daghighi, 2015; Rehman, Sun, Wah, & Khan, 2017).

2. Considering the variations in computational capabilities and battery power resources in far-edge mobile devices, a rigorous analysis of MDSM algorithms has been performed. To this end, nine highly adopted algorithms were selected that were used for classification, clustering, and association rule mining in previous studies. The performance analysis was performed by varying size and speed of data streams. In addition, the performance of mobile devices was assessed to find the resource consumption for learning model generation, and early and late data fusion. The results of data mining algorithms were also assessed to find the difference in device-based and desktop-based generation of learning models (Rehman, Liew, & Wah, 2014a).
3. A component-based architecture for distributed MDSM applications in MECC systems has been proposed. The architecture is based on three layers of executions namely, *a*) local analytics (LA) layer whereby the components for MDSM applications as well as supporting components for transient data management, resource profiling, context collection, and data offloading run inside far-edge mobile devices, *b*) collaborative analytics (CA) layer whereby data stream mining components execute at mobile edge servers. In addition the operational components for ad hoc network formation, knowledge synchronization, and garbage collection run in mobile edge servers, and *c*) cloud enabled analytics (CLA) layer whereby the CC servers provide application clones of MDSM applications (Rehman, Liew, & Wah, 2014b).

4. A dynamic application execution model has been proposed whereby the application components dynamically run in far-edge mobile devices, mobile edge servers, and CC servers. The conventional online data stream mining process was modified in order to enable dynamic application execution and handle continuously streaming data. The model uses mobile devices as primary platform in order to execute MDSM applications. The execution model runs certain application components, such as the components for data acquisition, data adaptation, data preprocessing, and data fusion, strictly inside mobile devices. However it dynamically offloads the data stream in mobile edge servers and CC servers after performing cost benefit analysis by considering current workloads, available and estimated computational, memory, and energy resources, and device usage context information. The performance analysis of execution model was performed by recruiting 12 graduate students from University of Malaya, Malaysia. To this end, a real-time online mobile activity detection application was developed and deployed on the students' mobile devices. The reported experimental outcomes were based on 15 days of experimentation that resulted in around five million feature vectors, with each feature vector representing a physical activity of a user. The results showed that dynamic execution model enabled maximum data processing in mobile devices which resulted in energy and memory efficiency as well as reduced data transmission in MECC systems (Rehman, Liew, et al., 2016).
5. An adaptive application execution model has been proposed in order to adaptively switch execution behavior of MDSM applications in MECC systems. The model enables multi-point transient data stream management and multi-point data stream offloading schemes and enables resource profiling and context collection components at each stage of application execution. The model enables resource profilers and resource estimation components to profile the execution behavior of applica-

tion components at each stage of execution. The performance of adaptive execution model was evaluated by continuously running mobile activity detection applications on users' mobile phones. The reported results show that adaptive execution model helped in reducing makespan almost by half of which was experienced in dynamic execution model. In addition, the adaptive model outperformed the static execution models in terms of battery power, memory and bandwidth consumption.

1.5 Thesis Outline

Chapter 2 discusses the state of the art execution models for MDSM applications. The chapter is organized as follows. First of all, the chapter presents the anatomy of MDSM applications. To familiarize with MDSM platforms, a detailed discussion on different topological settings and associated opportunities and challenges is made. The review of existing literature about execution models is analyzed and synthesized. The review is made in terms of model types, granularity of data processing, and behavior of models. In addition, existing MDSM platforms were reviewed in terms of data management and adaptation strategies, application partitioning schemes, and computation offloading and data transfer methods. In order to find the research problem and scope of the thesis, the chapter discusses critical factors of complexity in MDSM applications and articulates the research gap to control application complexities at platform level.

Chapter 3 is divided into two parts. In the fist part, a detailed feasibility and performance analysis of selected data mining algorithms has been presented. To this end, MDSM algorithms were assessed in terms of critical factors of complexity which were articulated in Chapter 2. In the second part, a framework for distributed MDSM applications has been presented. In addition, the component-based three-tier architecture has been presented with detailed discussion on operations performed at each layer. Finally, formal verification, modeling and validation of proposed architecture has been performed.

At the end, the state explosion problem in proposed architecture has been discussed.

Chapter 4 presents the MDSM application architecture and state transition model. In addition, the chapter highlights compute-intensive and data-intensive applications and discusses the possibilities of application execution in MECC systems. This chapter also presents the proposed dynamic and adaptive execution models that are based on single point and multi-point data stream management and data stream offloading schemes.

Chapter 5 presents the performance evaluation of execution models. Firstly, the chapter presents the performance results of MDSM applications with static execution models in standalone far-edge mobile devices, and far-edge to far-edge (F2F) and far-edge to cloud (F2C) settings. In addition, the chapter presents the correlation analysis of memory utilization, makespan, and battery power consumption in static execution models. Secondly, the chapter presents performance evaluation of MDSM applications using dynamic execution model followed by performance evaluation of application execution using adaptive execution model. Furthermore, the chapter presents comparison of proposed dynamic and adaptive execution models with static execution models and qualitatively compares existing MDSM platform with proposed UniMiner architecture.

Chapter 6 highlights the achievements of this research work. In addition, it presents the future research agenda by highlighting the research issues and future research areas for the applicability of this research work.

CHAPTER 2: EXECUTION MODELS FOR MDSM PLATFORMS

"...the execution model specifies how application components are executed..."

Briam W. Kernighan and Dennis M. Ritchie (February 1978)

Chapter 2 outlines the anatomy of MDSM applications in Section 2.1 and presents a detailed discussion of different topological settings for MDSM platforms in Section 2.2. In Section 2.3, the chapter presents discussion on execution models in different perspectives including model type, data processing granularity and model behavior. In addition, the review of data management and adaptation strategies, application partitioning schemes, computation offloading methods, and data transfer schemes is presented. Section 2.4 presents critical factors of complexity in MDSM applications when executed in far-edge mobile devices. Section 2.5 presents the gap analysis of existing literature and perceived solutions to control the complexity. Finally, the chapter is summarized in Section 2.6.

2.1 The Anatomy of MDSM Applications

The MDSM applications are based on multistage execution process whereby the data streams are transformed from raw data into knowledge patterns. The MDSM application components are executed at six stages (see Figure 2.1) whereby a huge amount of heterogeneity is present at each stage. The detailed taxonomy and literature review of methods handling heterogeneity at these stages is presented in Appendix A for interested readers. Following operations are performed at each stage:

- The application components at S1 collect data stream from data source(s) which reside either onboard in far-edge mobile devices or off-board in external environments.
- The data fusion operations are performed by application components at S2 whereby data streams from one are more data sources are integrated for data preparation.

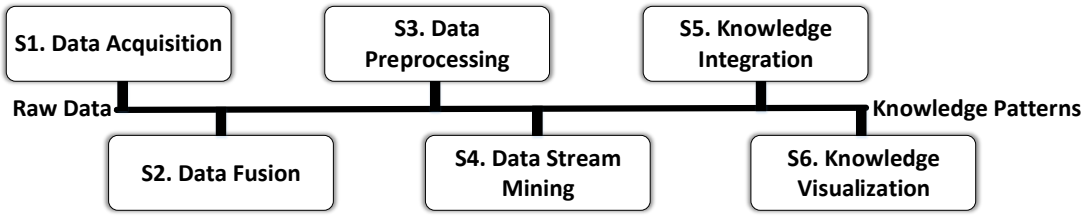


Figure 2.1: Multistage MDSM Applications.

- At S3, the data preprocessing components execute data filtration, noise removal, feature extraction and reduction, and outliers detection methods.
- The data stream mining components at S4 enable learning models generation schemes and execute model-based and model-less data mining algorithms.
- The discovered knowledge patterns are integrated, summarized, and stored by application comportments at S5.
- Finally, at S6, the application components execute knowledge management and visualization operations.

Considering the anatomy of MDSM applications and the literature review in Appendix-A it was found that MDSM application components need variable computational resources to perform seamless knowledge discovery operations. A variety of MDSM platforms were designed to provide required computational resources but the computing devices and systems also involve massive heterogeneity (Abdallah, Gaber, Srinivasan, & Krishnaswamy, 2015; Gaber, Stahl, & Gomes, 2014; P. D. Haghighi et al., 2013; Jayaraman, Gomes, et al., 2014). Therefore, designing an efficient execution model is a challenging task. To handle platform level heterogeneity, Section 2.2 defines different MDSM platforms and highlights the opportunities and challenges. Section 2.3 presents the literature review of execution models in different platforms.

2.2 Topological Settings of MDSM Platforms

The MDSM platforms were deployed in multiple topological settings (Abdallah et al., 2015; Gaber, Stahl, & Gomes, 2014; P. D. Haghighi et al., 2013; Jayaraman, Gomes, et al., 2014). The underlying communication models include multi-granularity computing devices and systems including mobile devices, Internet and intranet based application servers and cloud data centers to name a few (Jayaraman, Perera, Georgakopoulos, & Zaslavsky, 2014; Kargupta et al., 2010).

2.2.1 Far-edge Mobile Devices

Far-edge mobile devices are defined as any portable system or device with wireless communication interfaces and ability to produce or process data. Smartphones, wearable sensors, wireless body sensor networks, smart vehicles, and mobile Internet of Things are a few examples of far-edge mobile devices. Although modern far-edge mobile devices enable rich mobile streaming data applications such as virtual reality, computer vision, and multimedia applications using cloud augmented computational resources (Satyanarayanan et al., 2015) however the execution of heterogeneous MDSM applications inside far-edge devices is a challenging task (Rehman et al., 2015). Far-edge mobile devices usu-

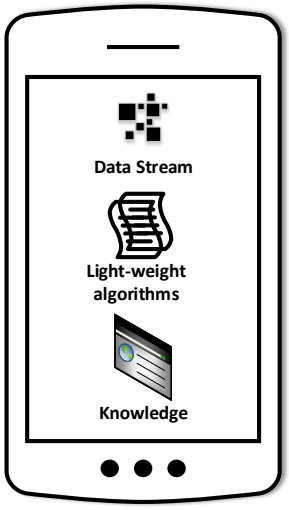


Figure 2.2: MDSM Application Execution in Far-edge Mobile Devices.

ally contain limited computational resources and battery power therefore MDSM applications consider these limitations for efficient process execution in mobile environments (Krishnaswamy, Gama, & Gaber, 2012). Data stream mining components, as shown in Figure 2.2, are designed to be light-weight to unleash the maximum utilization of onboard computational resources.

Opportunities: The deployment of MDSM applications in far-edge mobile devices creates multifold opportunities. These applications help in reducing outgoing data streams which in turn reduce network traffic as well as minimize cost of communication in terms of bandwidth utilization and GSM data plans (Jayaraman, Perera, et al., 2014). In addition, the close proximity of data sources and computational components in far-edge mobile devices lowers the makespan when compared with offloading raw data streams in external environments such as application servers, cloud data centers, and computing grids (Jayaraman, Gomes, et al., 2014). The privacy preservation and local knowledge availability are additional benefits of the deployment of MDSM applications in far-edge mobile devices (Arunkumar, Srivatsa, & Rajarajan, 2015). In summary, the knowledge patterns acquired after onboard execution of MDSM applications enable local knowledge availability, reduce dependency over external systems for data processing, and preserve privacy of users' personal data (Jayaraman, Perera, et al., 2014).

Challenges: The limited computational resources such as memory and CPU, and battery power are the main bottlenecks in far-edge mobile devices (Krishnaswamy et al., 2012). The challenge arises due to miniaturization of computational elements and the constraints of small size, light-weight, and less heat dissipating devices. Although modern far-edge mobile devices come with sophisticated computational elements and enable power saving functions however the deployment of heterogeneous MDSM applications is still a challenging task (Gaber, Gama, et al., 2014). Existing MDSM platforms adapt the execution behavior according to resource availability in terms of memory, battery power,

and CPU power and situation awareness which enforce light-weight execution and results in compromising the quality of knowledge patterns (P. D. Haghighi et al., 2013).

2.2.2 Far-edge to Far-edge Communication Model

Far-edge to Far-edge (F2F) communication model is defined as a set of Far-edge devices that can communicate among each other directly without any external controlling mechanism. For example, F2F communication model facilitates in direct communication between smart watch like Samsung Gear and a Smartphone such as Samsung Galaxy S5 (Samsung, 2014) (see Figure 2.3). Similarly multiple devices owned by a single user such as wearable devices, smartphones, tablet PCs, and laptops can establish a direct connection through Bluetooth, Zigbee, and Wi-Fi interfaces (Al-Fuqaha et al., 2015). The F2F communication model is adequate for single-user multi-device settings where far-edge devices can initiate point-to-point and group communication sessions to execute MDSM applications collaboratively (Framework, 2015).

Opportunities: In F2F settings, the closer far-edge mobile devices can pool the computational resources to augment the resource-constrained far-edge mobile devices with maximum execution support within Personal Area Network (PAN) (L. Wang, Gu, Tao,

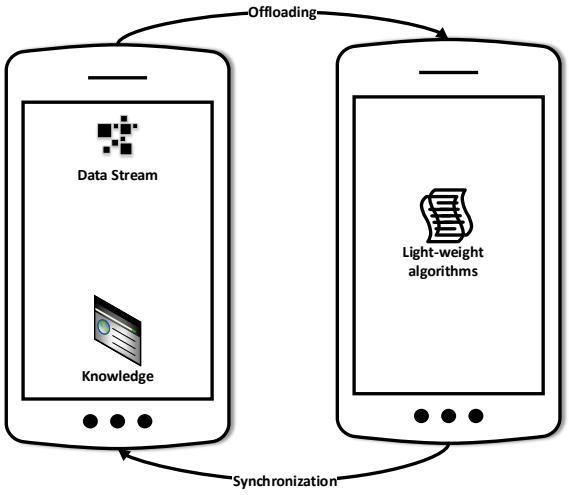


Figure 2.3: MDSM Application Execution using F2F Communication Model.

& Lu, 2012; Rehman et al., 2015). In addition, the F2F settings enable to distribute application logic among different far-edge mobile devices for seamless application execution (Li, Peng, Xiao, & Hua, 2015; L. Wang et al., 2012). For example, the far-edge mobile devices with minute computational facilities perform data acquisition operations and facilitate in data transfer operations in relatively high-power far-edge mobile devices. The high-power far-edge mobile devices enable data stream mining operations and synchronize the knowledge among other far-edge mobile devices in PAN (Gu, Wang, Wu, Tao, & Lu, 2011).

Challenges: The distribution of application logic is a major challenge in F2F settings (L. Wang et al., 2012). The MDSM applications need to be carefully designed to run the resource-intensive components in relevantly high-power far-edge mobile devices in order to avoid resource unavailability in low-power far-edge mobile devices (Min & Cho, 2011). The static application logic distribution in F2F settings may introduce high coupling among several far-edge mobile devices which affect the resource provisioning from other resource-constrained far-edge mobile devices; especially when a far-edge mobile device executes resource-intensive components but do not possess sufficient resources at a particular time (Braojos, Beretta, Constantin, Burg, & Atienza, 2014). Keeping track record of mobility patterns for adaptive application execution could also become a challenging task in F2F settings because the far-edge mobile devices are assumed to be in locality for seamless application execution.

2.2.3 Mobile Edge Servers

Mobile Edge server is defined as any mobile device or mobile system that resides at one-hop wireless distance from far-edge mobile devices. A mobile edge server enables data stream mining functionality by providing mobile services to thin and thick far-edge mobile devices (Gaber, Stahl, & Gomes, 2014) (see Figure 2.4). The thin far-edge mobile

devices function as data acquisition and data transfer elements however thick far-edge devices enable additional functionality of executing light-weight data stream mining algorithms. Some examples of mobile edge servers include frequently co-located far-edge devices such as personal mobile devices (*i.e.* wearable devices, smartphones, Tablet PCs, and laptop computers), far-edge mobile devices owned by co-workers, family members, and friends, and shared far-edge mobile devices such as appliances in smart home environments, and office equipment in smart co-working spaces.

Opportunities: The co-location and co-movement of far-edge mobile devices and mobile edge servers reduce dependency over large-scale centralized systems (Kargupta et al., 2010). In addition, far-edge mobile devices can offload resource-intensive tasks to mobile edge servers without Internet connections by utilizing local communication channels (Gaber, Stahl, & Gomes, 2014). Mobile edge servers may owned and controlled by different users therefore MDSM applications should be device-centric and mobile edge servers should provide complete application clones to reduce the high coupling. An added advantage of mobile edge servers is the elastic service availability where far-edge mobile devices can offload data mining tasks in multiple mobile edge servers using device-centric task scheduling schemes. The addition of location aware context features in MDSM can enable mobile distributed intelligence where multiple far-edge mobile devices can sense and log the data and act like both as standalone far-edge mobile devices and as mobile

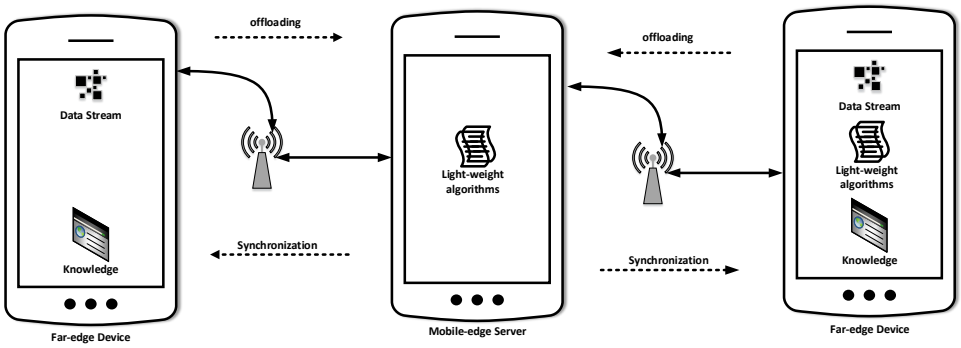


Figure 2.4: MDSM Application Execution using Mobile Edge Servers.

edge servers.

Challenges: The resource constraints in mobile edge servers are still a challenge which enforces the light-weight service provision to MDSM applications. In addition, the mobility of devices may degrade the performance of MDSM applications. The privacy preservation of users' personal data is another challenge that mobile edge servers should consider. Moreover, variety in operating systems of mobile edge servers, programming environments, and communication interfaces requires extensive profiling of mobile edge servers to provide optimal user experience.

2.2.4 Immobile Edge Servers

Immobile edge servers are defined as the physically static and resourceful computing systems that reside at one-hop wireless distances from far-edge mobile devices. The immobile edge servers include cloudlets, micro data centers, radio access network (RAN) servers in GSM networks, application servers, and smart-routers in local area networks to name a few (Bonomi et al., 2012; Satyanarayanan et al., 2015; Bahl, 2015; Ha & Satyanarayanan, 2015). Similar to mobile edge servers, the communication model facilitates thin/thick far-edge devices but physically bounded nature of immobile edge servers enforce collaborative execution models between far-edge mobile devices and immobile edge servers (Ferreira, Duarte, & Preguiça, 2010) (see Figure 2.5). The collaborative execution model need to perform operational monitoring at far-edge mobile devices and immobile edge servers for seamless execution of MDSM applications (Sherchan et al., 2012).

Opportunities: The deployment of MDSM applications at immobile edge servers helps in prolonging battery lifetime of far-edge mobile devices (Satyanarayanan et al., 2015). In addition the availability of high computational resources reduces the application processing time hence minimizes the makespan (Bahl, 2015).

Challenges: Despite of enhanced application performance, the immobile edge servers

need to address few challenges such as profiling mobility of far-edge mobile devices to maximize resource provisioning (A. Ahmad & Ahmad, 2016), enabling live virtual machine migration for seamless application execution (R. W. Ahmad et al., 2015), saving and resuming application states to seamlessly switch among different networks (Ha & Satyanarayanan, 2015), enabling application partitioning schemes to statically and dynamically partition MDSM applications in locally runnable components and offloadable components (J. Liu et al., 2015), and performing cost-benefit analysis for the favorability of offloading certain computations in immobile edge servers (M. A. Khan, 2015).

2.2.5 Mobile Cloud Computing System

Mobile cloud computing (MCC) systems are defined as the computing systems that provide heterogeneous computing, networking, and storage services to far-edge mobile devices through large scale data centers. The application models for MCC based MDSM applications involve thin and thick far-edge mobile devices (Altomare, Cesario, Comito, Marozzo, & Talia, 2013) (see Figure 2.6). For example, far-edge devices directly upload data stream in cloud data centers and data stream mining operations are performed in cloud environments (Talia & Trunfio, 2010). Alternately, far-edge mobile devices perform data stream mining operations locally using onboard computational resources and enable on-demand data offloading when required (Jayaraman, Gomes, et al., 2014).

Opportunities: The MCC systems offer many opportunities to augment MDSM ap-

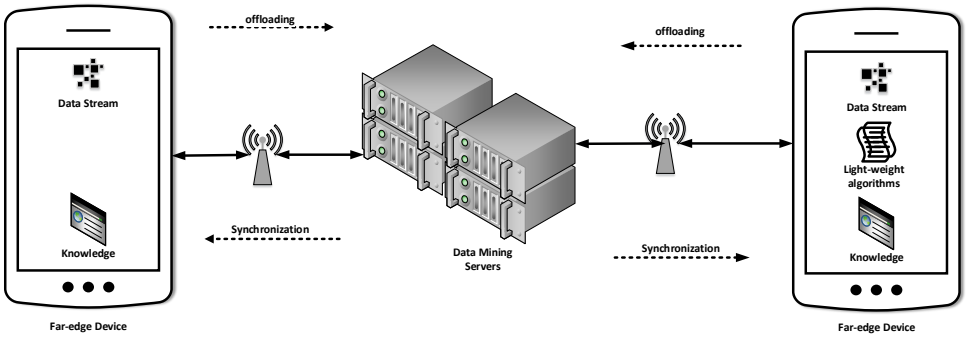


Figure 2.5: MDSM Application Execution using Immobile Edge Servers.

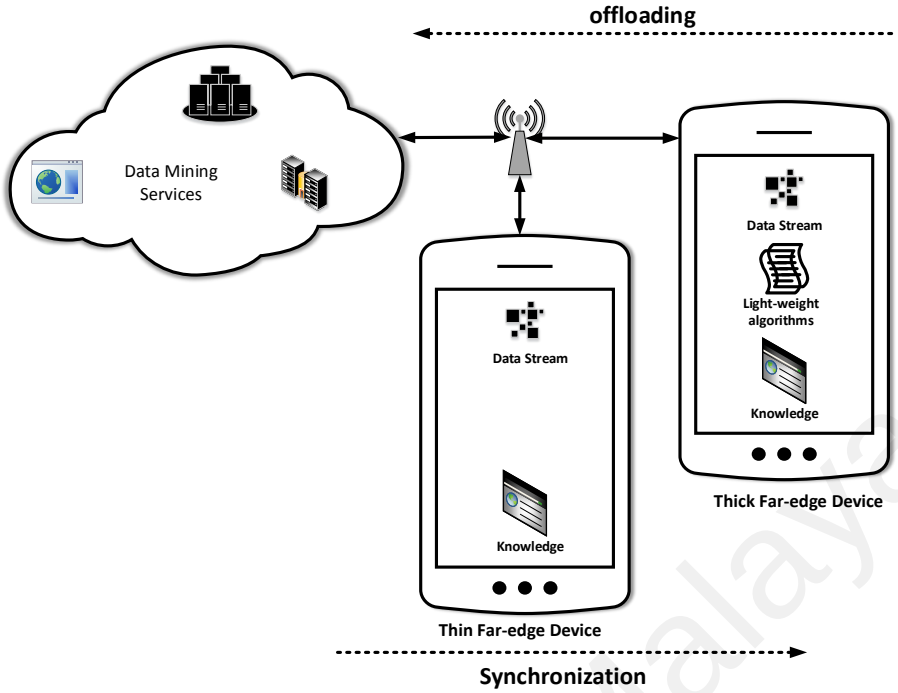


Figure 2.6: MDSM Application Execution using Mobile Cloud Systems.

plications. The MCC systems ensure provisioning of highly available and hypothetically unlimited computing, networking, and storage resources through large-scale data centers. MCC systems enable multiple form of services namely Storage-as-a-Services (SaaS), Application-as-a-Services (AaaS), Network-as-a-Services (NaaS), and a large plethora of services at hardware, operating systems, and application levels (Fernando et al., 2013).

Challenges: Ranging from cluster level and data center level issues to operating systems, and application level issues, the MCC systems face many challenges for MDSM applications. The dependency on the availability of Internet connections, high data rates due to raw data transfer, increased makespan due to multi-hop distance between far-edge mobile devices and cloud data centers, resource provisioning, on-demand load-balancing, seamless application execution, adaptive and dynamic computation offloading, the deployment of data stream mining services and updating learning models, and increasing privacy concerns are a few challenges that MCC systems need to handle for the deployment of MDSM applications (Kumar, Liu, Lu, & Bhargava, 2013; Yuan & Herbert, 2014).

2.3 MDSM Execution Models

This section presents the review of highly relevant methods and schemes that are used to address the execution model level issues. The execution models were studied in multiple perspectives such as model type and behavior and the granularity of data processing. In addition, existing studies were analyzed in terms of data management schemes, application partitioning methods, adaptation strategies, and data transfer and computation offloading schemes. Since the scope of this thesis is limited to MDSM application execution, therefore, the conventional large-scale stream execution models were not analyzed and presented in this thesis. The discussion on conventional stream execution models were presented and published in our previous study (Rehman et al., 2015) hence it is not covered in the scope of this thesis. The studied execution models in this thesis enable the stream executions in one of above mentioned topological settings. Literature review of existing MDSM platforms is presented in terms of topological setting, model type, data processing granularity, and model behavior in Table 2.1.

2.3.1 Model Type

Execution models for MDSM applications either remain standalone or distributed among multiple devices and systems. The standalone execution models execute all application components in a single device or systems (Krishnaswamy et al., 2009). Historically, the standalone models were used with far-edge device based systems whereby all application components from data acquisition stage to knowledge visualization stage are executed using onboard computational and battery resources (J. B. Gomes, Krishnaswamy, Gaber, Sousa, & Menasalvas, 2012a; Abdallah et al., 2015). The execution of MDSM applications in resource-constrained environments is a major bottleneck in the performance of standalone models (Krishnaswamy et al., 2012; Gama, 2013). To address this challenge, existing systems perform partial application execution in far-edge devices whereby cer-

tain computation and energy-intensive operations such as learning model generation are performed in other devices and systems (Yuan & Herbert, 2014). The resultant models are transferred in far-edge device for knowledge discovery. Alternatively, the MDSM applications perform passive learning whereby one-time training is performed (Siirtola & Roning, 2013), or the MDSM applications use active learning methods within limited search spaces (Yuan & Herbert, 2014; Oneto, Ghio, Ridella, & Anguita, 2015). The challenge of resource limitation in far-edge mobile devices could be addressed by adjusting few parameters in the data stream such as size of data, number of variables, and size of results produced by data mining algorithms (Comito & Talia, 2013). The adaptation strategies are another alternate to handle resource constraints whereby the MDSM applications adapt the execution behaviors according to onboard available resources. The discussion on adaptation strategies is presented later in this section. Since standalone execution models need to compromise on quality of knowledge which is reduced due to parameter tuning and adaptation strategies (Krishnaswamy et al., 2009). The distributed application execution using nearby far-edge mobile devices becomes a feasible choice.

The distributed execution models are either collaborative or non-collaborative. The collaborative execution models involve multiple devices and systems to facilitate application execution whereby application components exchange certain information about underlying devices and platforms prior to data processing. To this end, existing systems, such as Pocket Data Mining (PDM), use agent-oriented architecture to enable collaborative execution model in distributed F2F environments (Gaber, Stahl, & Gomes, 2014). The far-edge device sends mobile agent for data mining in other devices which collect the information about available data sources and available implementations of data mining algorithms in other mobile devices. The decision for collaboration is made by matching the required data sources and resources. The agent-oriented architecture benefits by enabling a decentralized execution model and reduced communication within the ad hoc

F2F network. However, continuous running of mobile agent leads to incremental data communication and extra battery power consumption.

The addition of context-aware features enhances the performance of collaborative execution models whereby the MDSM applications operating in same locality with similar contextual information can handle concept drift (*i.e.* detecting changes in data stream) very effectively. The context aware collaborative execution model in F2F settings helps in data reduction, communication efficiency and improved data mining results. In context of collaborative execution model in F2F settings, a low level implementation of distributed k-means clustering algorithms was made by researchers (Ortiz et al., 2015). The algorithm was combined with contextual information and approximation techniques and the implementation reduced the makespan from almost two hours to 657 seconds. However, such implementations needs a lot of programming efforts to develop MDSM applications. The collection of contextual information at cloud side helps in effective collaboration in far-edge to cloud (F2C) settings (Sherchan et al., 2012). In such execution model, far-edge devices collect contextual information and transmit to cloud servers which infer the relevant contexts and suggest new configurations. The MDSM applications at far-edge mobile devices reconfigure the behavior accordingly (Jayaraman et al., 2012). The cloud based context awareness improves the performance of MDSM applications but the increased dependency over persistent Internet connections is a major bottleneck in such applications. The collaborative execution models in F2F and F2C settings were also used for opportunistic sensing and light-weight data processing in far-edge mobile devices. MOSDEN is an example of such platform whereby the remote servers delegate the sensing and data processing tasks to far-edge mobile devices where light-weight MDSM applications perform the required tasks and the results are communicated back to MOSDEN servers (Jayaraman, Perera, et al., 2014). However, these models also require persistent Internet connections for seamless application execution.

Unlike collaborative execution models, the application components in non-collaborative execution models just perform the knowledge discovery and data transfer processes without consulting the remote devices and servers. For example, MobiSens performs data acquisition and activity segmentations at far-edge mobile devices (Wu, Zhu, & Zhang, 2013). The collected data streams are transferred to remote severs in case the devices are connected with Internet and being charged. The remote servers perform knowledge discovery operations and resultant knowledge is used by remote applications deployed on back-end server. Similarly, a service-oriented architecture was proposed by researchers whereby far-edge mobile devices just perform data acquisition and transmit the data stream to remote servers whereby the required data mining tasks are performed and results are communicated back to far-edge mobile devices (Talia & Trunfio, 2010). Such execution models have multiple issues including increasing bandwidth utilization cost, extra energy consumption for data communication, requirement of persistent Internet connectivity, and high-coupling between application components.

Irrespective of execution model, the training of learning models is a computationally expensive task therefore it becomes very hard to train the learning models onboard in far-edge mobile devices. CARA implements a cloud based learning module for MDSM applications (Yuan & Herbert, 2014). The system works by downloading initial learning model in far-edge mobile devices. The application components at far-edge device collect and process data streams for knowledge discovery and assess the quality of knowledge patterns. In case of newly found patterns, the results are communicated back to cloud servers whereby the learning models are updated accordingly. Although this strategy enables the execution model to address the issue of resource constraints however the cost of data communication may drastically increase when MDSM applications are executed with continuously evolving data streams (Yuan & Herbert, 2014).

2.3.2 Granularity of Data Processing

The data processing behavior in existing execution models vary due to massive heterogeneity at application and platform levels.

Considering resource limitations in far-edge mobile devices, the MDSM application components are made light-weight by reducing the functionality of data processing (Gaber, 2009; Krishnaswamy et al., 2009), or the amount of data is reduced (Comito & Talia, 2013). The light-weight components with reduced functionality help in achieving data processing goal using far-edge devices but the MDSM applications may compromise on the quality of knowledge patterns (P. Haghighi, Gaber, Krishnaswamy, & Zaslavsky, 2007). On the other hand, data stream reduction may result in data loss which results in loosing information fidelity (Comito & Talia, 2013). Alternatively, heavy-weight data processing is performed using large-scale computing infrastructures such as LAN servers, computing clusters, and large-scale data centers (Kargupta et al., 2010; Talia & Trunfio, 2010). During heavy-weight data processing, the application components provide maximum functionalities without considering underlying computational resources.

Alternately, the MDSM applications perform partial data processing in F2F and F2C settings (Jayaraman, Gomes, et al., 2014; Ortiz et al., 2015). To this end, far-edge devices execute light-weight application components for initial data processing and the resultant data or knowledge patterns are aggregated in remote servers for further analysis and executions. In few platforms, the partial data processing is performed by running only one or two components in far-edge devices while rest of the executions are performed in remote servers. For example, MobiSens performs data acquisition, aggregation, and filtration in far-edge devices while rest of the application is executed in cloud servers (Wu et al., 2013). Similarly, in collaborative F2F settings, the MDSM applications perform data ac-

Reference	Topology	Standalone	Collaborative	Distributed	LW	HW	Partial	Static	Dynamic	Adaptive
(Krishnaswamy et al., 2009)	Far-edge	Yes	No	No	Yes	No	No	No	Yes	Yes
(Kargupta et al., 2010)	F2C	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No
(Gu et al., 2011)	F2F	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No
(Sherchan et al., 2012)	F2C	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes
(P. Liu, Chen, Tang, & Yue, 2012)	Far-edge	Yes	No	No	No	Yes	No	Yes	No	No
(Wu et al., 2013)	F2C	No	Yes	Yes	Yes	Yes	No	Yes	No	No
(Yuan & Herbert, 2014)	F2C	No	Yes	Yes	No	Yes	No	Yes	No	No
(Yang et al., 2014)	Far-edge	Yes	No	No	No	Yes	No	No	Yes	No
(Srinivasan et al., 2014)	Far-edge	Yes	No	No	No	Yes	No	Yes	No	No
(Gaber, Stahl, & Gomes, 2014)	F2F	No	Yes	Yes	Yes	No	No	No	Yes	Yes
(Jayaraman, Perera, et al., 2014)	F2C	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No
(Jayaraman, Gomes, et al., 2014)	F2C	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
(Abdallah et al., 2015)	Far-edge	Yes	No	No	Yes	No	No	No	Yes	Yes

Table 2.1: Execution Models for MDSM Applications.

quisition and feature extraction in far-edge devices and remaining components are executed in remote servers. Partial data processing in far-edge mobile devices is a feasible solutions but distributed application logic introduces high coupling among application components. The MDSM applications may fail when the Internet connections are unavailable or the far-edge devices are roaming among multiple communication networks.

2.3.3 Model Behavior

The behavior of execution models differs in terms of static, dynamic, and adaptive operations. In static execution models, the application components run without considering resource availabilities and execution workloads (P. Liu et al., 2012; Yuan & Herbert, 2014; Srinivasan et al., 2014). On the other hand, dynamic execution models consider the execution workloads and search for required resources before application execution (Krishnaswamy et al., 2009; Abdallah et al., 2015). The adaptive execution models add the adaptation mechanism in application execution processes in order to perform optimal application execution (Gaber, 2009; Jayaraman, Gomes, et al., 2014). Existing platforms perform static executions in F2F and F2C settings. However, for application execution in far-edge mobile devices, a few studies were proposed whereby the MDSM applications dynamically and adaptively execute using onboard computational resources by switching among light-weight and heavy-weight data processing.

2.3.4 Data Management and Adaptation Strategies

MDSM applications are bounded to operate on continuously streaming data; far-edge devices need to consider available memory resources in order to efficiently execute application components. To handle this issue, MDSM applications either temporarily store the collected data stream using onboard storage and memory buffers or directly execute data streams by optimizing the available resources using adaptation strategies (Kargupta et al., 2010; Wu et al., 2013; Jayaraman, Perera, et al., 2014; Sherchan et al., 2012).

Although most of the existing execution models perform direct in-memory operations, few platforms use data management operations. For example MobiSens performs data acquisition in local storage and transfers collected data streams to cloud data centers when the devices are connected through Wi-Fi and are in charging mode (Wu et al., 2013). Similarly, CARDAP and MOSDEN performs data management by performing data reduction using light-weight application components in far-edge devices and storing resultant patterns in local storage (Jayaraman, Perera, et al., 2014; Jayaraman, Gomes, et al., 2014). The systems also enable query manager components which perform on-demand data offloading when requested by cloud servers. CAROMM performs on-the-move mining whereby it uses light-weight adaptive clustering components over data streams to detect changes in data streams (Sherchan et al., 2012). CAROMM uploads the data streams after observing significant changes. MineFleet is one of pioneer MDSM platforms and it stores the data streams after performing data analysis (Kargupta et al., 2010). MineFleet clients upload locally stored data streams to MineFleet servers whenever the connectivity is available. Star, a recent framework, stores results of MDSM applications and immediately discards raw data streams after processing (Abdallah et al., 2015).

Table 2.2 presents a detailed literature review of proposed adaptation strategies. The adaptations are made at system level to adapt the generic processing behavior of MDSM applications (Pasricha, Donohoo, & Ohlsen, 2015). Alternately, the adaptation strategies work at algorithm level by altering the execution behaviors of data stream mining algorithms (Abdallah et al., 2015). The adaptations are made using multiple parameters such as data rate, memory, CPU, context aware features, learning models, or any specific event. The data rate based adaptive strategies work by monitoring the velocity of incoming and outgoing data streams (P. D. Haghighi et al., 2013). The adaptive data stream mining algorithms adjusts the execution behavior according to data rates. The memory and CPU

Reference	Platform	Algorithm	Data Rate	CPU	Memory	Context	Learning Model	Event
(Pasricha et al., 2015)	Yes	No	No	Yes	No	No	Yes	Yes
(Abdallah et al., 2015)	No	Yes	No	No	No	No	Yes	No
(P. D. Haghighi et al., 2013)	Yes	Yes	Yes	Yes	Yes	Yes	No	No
(Lu et al., 2012)	No	No	No	No	No	No	Yes	No
(Gaber, Stahl, & Gomes, 2014)	No	Yes	Yes	Yes	Yes	No	No	No
(Stahl et al., 2012)	No	Yes	Yes	Yes	Yes	No	No	No
(Jayaraman, Perera, et al., 2014)	Yes	Yes	Yes	Yes	Yes	No	No	No
(Eom, Figueiredo, Cai, Zhang, & Huang, 2015)	No	No	No	No	No	No	Yes	No
(Sherchan et al., 2012)	Yes	Yes	Yes	Yes	Yes	Yes	No	No
(Jayaraman, Gomes, et al., 2014)	Yes	Yes	Yes	Yes	Yes	Yes	No	No
(Yuan & Herbert, 2014)	Yes	No	No	No	No	No	Yes	No
(Kargupta et al., 2010)	No	Yes	No	No	No	No	No	No

Table 2.2: Adaptation Strategies.

based adaptation strategies works by profiling the computational requirements of data stream mining algorithms and adjusting the execution behavior accordingly. The context-aware adaptive strategies profile different situations and adjust the execution behavior of MDSM platforms whenever a relevant situation is inferred (Sherchan et al., 2012). The learning model based strategies consider the execution history of MDSM applications, learn the execution patterns, and alter the execution behavior according to predicted settings (Lu et al., 2012; Yuan & Herbert, 2014). Event based strategies work by adopting the execution behavior of data stream mining algorithms accordingly when a specific event occurs (Pasricha et al., 2015).

2.3.5 Application Partitioning

Distributed MDSM applications are partitioned to run in multiple devices and systems. The application partitioning strategies are controlled by either far-edge devices, cloud servers, or edge servers. MDSM applications are either partitioned dynamically at runtime after assessing the resource requirements of the running processes or the application are partitioned in fixed form where specific application components run at designated devices and systems (Gaber, Stahl, & Gomes, 2014; Ortiz et al., 2015). Table 2.3 presents a detailed literature review of application partitioning strategies in MDSM platforms. The applications are partitioned either on the basis of data or computations (L. Wang et al., 2012; Braojos et al., 2014). The data based application partitioning is performed by executing data parallel strategies where partial data streams are offloaded and executed in various devices and systems. The computation based partitioning is performed by measuring the computational requirement and granularity of data processing. In computation based partitioning partial tasks are executed in various device and systems and application partitioning is performed either offline or online. The offline partitioning is performed before or after the application execution however the online partitioning is performed during

Reference	Device	Cloud	Server	Edge	Type	Partitioning Mode	Model	Granularity	Form
(L. Wang et al., 2012)	Yes	NA	NA	NA	Offline	Data-based	Static	Data	Fixed
(Gaber, Stahl, & Gomes, 2014)	Yes	No	No	No	Offline	Data-based	Static	Data	Fixed
(Ortiz et al., 2015)	Yes	No	NA	NA	Offline	Data-based	Static	Data	Fixed
(Braojos et al., 2014)	Yes	No	No	No	Offline	Computation-based	Static	Learning Model	Fixed
(Min & Cho, 2011)	Yes	No	No	No	Offline	Data-based	Static	Data	Fixed
(Jayaraman, Perera, et al., 2014)	Yes	Yes	Yes	Yes	Offline	Data-based	Dynamic	Data	Fixed
(Dou et al., 2011)	Yes	No	No	No	NA	Data-based	Static	Data	Fixed
(Jayaraman, Gomes, et al., 2014)	Yes	Yes	No	No	Offline	Computation-based	Static	Application	Fixed
(Lin, Choy, Pang, & Ng, 2013)	No	No	Yes	No	Offline	Data-based	Static	Data	Fixed
(Yuan & Herbert, 2014)	Yes	Yes	Yes	NA	Offline	Data-based	Static	Data	Fixed
(Hassan, Wei, & Chen, 2015)	Yes	No	No	No	Online	Method-based	Dynamic	Method	Dynamic
(Talia & Trunfio, 2010)	No	No	Yes	No	Offline	Data-based	No	No	No
(Yoon, 2013)	Yes	Yes	NA	NA	Offline	Data-based	Static	Method	NA

Table 2.3: Application Partitioning Schemes.

the application execution process. Exiting literature still lacks the online schemes.

2.3.6 Computation Offloading

Existing computation offloading schemes are based on different communication models that vary in terms of client-server settings, virtual machine migration and mobile agent configurations (M. A. Khan, 2015). In client-server based settings, offloading components reside on the mobile device that offloads the computations after performing collaborative cost-benefit analysis for computation offloading favorability. Cost-benefit analysis is performed to label the local and remote computations for application partitioning (J. Liu et al., 2015) and resource-hungry computational tasks are offloaded to the nearest or designated surrogates (*i.e.* servers) in the cloud. The main concern with computation offloading in server-based MCC settings is the requirement for pre-installed mobile services in ad-hoc cloud environments. In virtual machine migration-based communication models, the memory image of a central cloud server is migrated in cloudlets, which lowers the communication cost as well as overall bandwidth utilization in highly-dense MCC environments (Satyanarayanan et al., 2009). Live virtual machine migration increases makespan in service provisioning (R. W. Ahmad et al., 2015). In addition, the preservation and resumption of application states during migration is also a major challenge (Satyanarayanan et al., 2009). In mobile agent communication models, the application clones are migrated in cloud environments to augment the mobile devices with cloud resources. However, mobile agent management and clone security are the main issues in mobile agent-based MCC environments (M. A. Khan, 2015). Table 2.4 presents the detailed literature review of computation offloading methodologies for MDSM platforms.

Computation offloading schemes function with single-site and multiple-site surrogate settings (Abolfazli, Sanaei, Ahmed, Gani, & Buyya, 2014). In case of single-site

surrogates, the application components are offloaded to the same server in the MCC architecture. However, this setting develops a tightly bounded relationship between mobile applications and their corresponding surrogates. Therefore, the dynamic mobility increases the makespan in distant mobile devices (Abolfazli et al., 2014). On the other hand, multiple-site surrogates work in two ways. Either application clones are provided at multiple sites using live virtual machine migration methods or different program components are executed at various surrogates. In addition, the virtual machine migration problem also brings the parallelization challenge, which needs to be addressed in multiple-site surrogates (Abolfazli et al., 2014). The programs should be effectively partitioned and mapped into graph data structures that are further optimized for seamless application execution in MCC environments. In addition, adaptive computation offloading schemes consider program execution contexts and previous program instances and devise optimal execution strategies accordingly. These schemes consider various parameters, including network connections and bandwidths, workloads, architectural heterogeneity and task deadlines. However, the decision of favorable offloading becomes complex due to varying bandwidth, resource availability and network dynamics (Abolfazli et al., 2014).

Computation offloading schemes in MCC environments are categorized as either static or dynamic (Kumar et al., 2013). In static schemes, one-time cost-benefit analysis is performed and offloading-favorable computations are offloaded in MCC environments. Dynamic computation offloading schemes initially perform cost-benefit analysis, implement online profiling, tag the offloadable program components during application execution, perform application partitioning for local and remote execution, and offload the computation offloading-favorable components in MCC environments (Kumar et al., 2013). Computation offloading is performed at different granularity levels. At the coarse-grained level, entire applications are offloaded in cloud servers. The coarse-grained level computation offloading is well-suited when cloud resources are available at one-hop

Reference	Mode	Type	Parameters	Offloading Devices	Offloading Servers
(L. Wang et al., 2012)	Offline	Static	NA	Single	Single
(Gaber, Stahl, & Gomes, 2014)	Offline	Static	NA	Single	Multiple
(Ortiz et al., 2015)	Online	Static	Makespan	Multiple	Multiple
(Stahl et al., 2012)	Offline	Static	NA	Single	Multiple
(Jayaraman, Perera, et al., 2014)	Both	Dynamic	Network Connection	Multiple	Multiple
(Eom et al., 2015)	Online	Dynamic	Classifier	Single	Multiple
(Sherchan et al., 2012)	Offline	Static	NA	Multiple	Single
(Jayaraman, Gomes, et al., 2014)	Offline	Static	NA	Multiple	Single
(Hassan et al., 2015)	Online	Dynamic	Multiple	Single	Multiple
(Talia & Trunfio, 2010)	No	No	NA	Multiple	Multiple
(Yoon, 2013)	Offline	Static	NA	Multiple	Single

Table 2.4: Computation Offloading Schemes.

distances from mobile devices. However, in case of cloudlets, live virtual machine migration may incur higher cost in terms of makespan. Alternately, the complete migration of entire application states in edge servers increases local computation costs. At fine-grained levels, computation offloading is performed at various application code levels, including method, task, object, thread, class and program levels. These different granularity levels increase the decision complexity of computation offloading. Optimal computation offloading strategies involve multiple offloading objectives such as performance enhancement, energy gain, reduced makespan, minimum bandwidth utilization cost.

2.3.7 Data Transfer Strategies

The MDSM applications transfer data streams among devices and systems in multiple ways. The simplest data stream transfer strategies are based on transferring raw data streams (Talia & Trunfio, 2010). The raw data streams are either stored onboard or directly collected from data sources. Sometimes MDSM applications perform initial data processing and transfer the intermediate data to other systems and sometimes the data stream mining algorithms are executed onboard in light-weight processing modes and resultant knowledge patterns are transferred to other devices and systems for aggregation and global knowledge view (Gaber, Stahl, & Gomes, 2014; Jayaraman, Perera, et al., 2014; Jayaraman, Gomes, et al., 2014). Table 2.5 presents the detailed literature review of data transfer methodologies for MDSM platforms.

The data streams are transferred in push-based, pull-based, on-demand, or opportunistic settings. In push-based strategies, the mobile devices simply transfer the data stream to connected devices and systems (Gaber, Stahl, & Gomes, 2014). In pull-based strategies, remote systems like cloud servers monitor the connections and periodically collect the data stream from mobile devices (Jayaraman, Perera, et al., 2014). The on-demand strategies work when the remote servers issue queries for data processing or

sensing to connected mobile devices which in turn perform the required operations and communicate the results back to requesting server (Jayaraman, Gomes, et al., 2014). On-demand data transfer strategies are useful for mobile crowd sensing applications. The opportunistic data transfer strategies monitor the connected devices and systems and find the feasible environment for pushing or pulling data streams among connected devices and systems (Hassan et al., 2015). Smart data reduction is another approach for data transfer where mobile devices perform the data stream mining operations and the results are communicated only if there is a significant change in the data stream (Jayaraman, Gomes, et al., 2014). Existing MDSM platforms primarily use push based data transfer schemes that need persistent Internet connections for effective communication. Therefore, efforts are needed to develop new strategies for pull based, on-demand, smart data reduction, and opportunistic data transfer.

Reference	Push	Pull	On-demand	Opportunistic	Smart DR
(L. Wang et al., 2012)	Yes	No	No	No	No
(Gaber, Stahl, & Gomes, 2014)	Yes	No	No	No	No
(Ortiz et al., 2015)	Yes	No	No	No	No
(Stahl et al., 2012)	Yes	No	No	No	No
(Jayaraman, Perera, et al., 2014)	Yes	Yes	Yes	No	No
(Eom et al., 2015)	Yes	No	No	No	No
(Sherchan et al., 2012)	Yes	No	No	No	No
(Jayaraman, Gomes, et al., 2014)	Yes	No	Yes	No	Yes
(Lin et al., 2013)	Yes	No	No	No	No
(Hassan et al., 2015)	Yes	No	No	Yes	No
(Talia & Trunfio, 2010)	Yes	No	No	No	No
(Yoon, 2013)	Yes	No	No	No	No

Table 2.5: Data Transfer Methods.

2.4 Critical Factors of Complexity in MDSM Applications

Referring to literature review in Appendix A, it was found that the complexity of MDSM applications in far-edge devices is affected by six critical factors (see Figure 2.7). The size of incoming data stream plays a vital role in application complexity and it affects the computational complexities of other application components at subsequent stages. Dur-

ing executions in far-edge mobile devices, existing methods use light-weight algorithms which do not consider the whole data stream therefore applications need to compromise on the quality of knowledge patterns. Alternatively, high data size directly impacts the performance of MDSM applications when executed with heavy-weight data stream mining algorithms. Likewise high data rate in MDSM applications increases the computational complexity. Exiting MDSM applications work online by performing in-memory operations with time constraints. The algorithms are executed as one-pass algorithms with the condition that current data stream must be processed before the arrival of next data stream. This condition impacts the performance of MDSM applications because during one-time processing, the applications can not perform iterative data processing hence produce inefficient results with historical data.

The choice of data fusion strategy helps in increasing or decreasing the computational complexity of MDSM applications. Early data fusion strategies produce redundant, noisy, and anomalous data streams because MDSM applications collect data streams without any filtration and/or preprocessing methods. On other hand, late and discriminatory data fusion strategies produce high quality data streams hence requires less computations at later stages. The operational behaviors such as populating data structures, the traversal methods, and nature of computational operations affect the computational complexity of data preprocessing and data mining algorithms. The computational complexity increases when the data stream mining algorithms are bounded to perform all operations using onboard computational resources. Existing systems use light-weight algorithms, that use shallow data structures to handle the computational complexity of data preprocessing and mining algorithms. Efforts are needed to deploy heavy-weight data stream mining algorithms with iterative processing behaviors and deep data structures in order to maximize the performance of MDSM applications.

The complexity of MDSM applications also increases during learning phase. On-

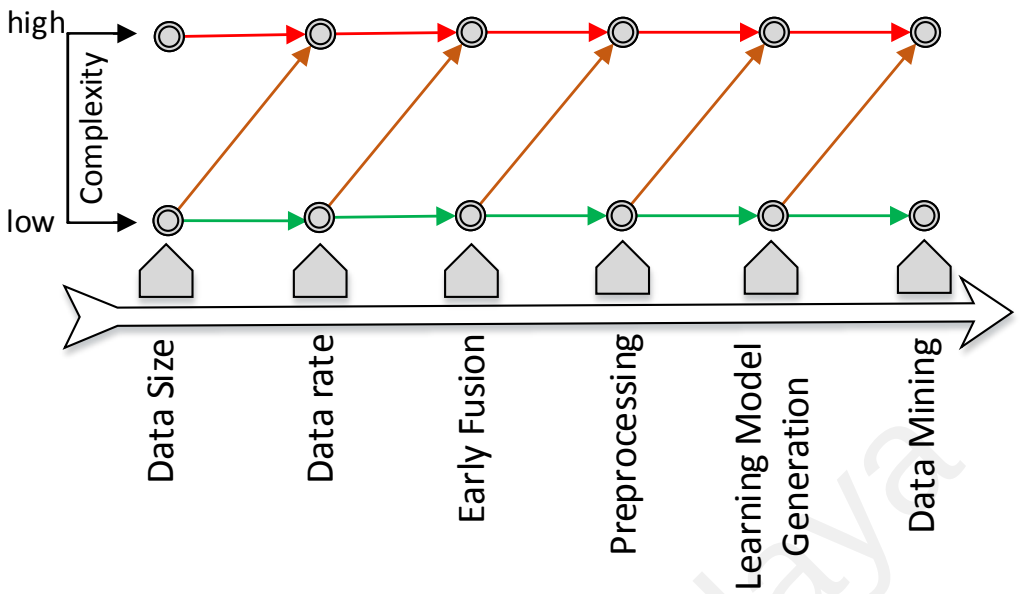


Figure 2.7: Factors Affecting Computational Complexity.

line learning over large streaming data becomes computationally infeasible due to resource limitations and constraint of keeping whole data stream in memory. The behavior of learning model such as supervised, unsupervised, and semi-supervised settings also affect the computational complexity of MDSM applications. The supervised and semi-supervised learning model initially uses labeled data stream hence learning algorithms are trained within a confined feature space. On the other hand, during unsupervised learning the leaning models need to be trained with high-dimensional complex data streams which quickly hamper the computational resources in far-edge mobile devices. In essence, high complexity in aforementioned critical factors impacts the MDSM applications as a whole. Therefore, increase in computational complexity at any stage impacts the subsequent operations in MDSM applications because the output data stream from one stage becomes the input of application components at next stage.

2.5 Controlling Complexity at Platform Level

Ideally, MDSM applications should perform maximum computational operations near the data sources in order to reduce makespan and bandwidth utilization cost. Existing execution models either work as standalone execution platforms in far-edge devices or distributed in F2F and F2C settings.

Due to resource limitations in far-edge mobile devices, the execution models support light-weight and adaptive data processing however compromise on quality of knowledge that is produced by MDSM applications. Alternately, when deployed in F2F settings, execution models need to perform device discovery and ad hoc network formation operations. Existing models enable agent-oriented architectures for device discovery which increase energy consumption due to roaming of multiple agents inside ad hoc network. In F2C settings, far-edge mobile devices continuously upload the data streams in cloud servers. This data communication increases bandwidth utilization cost in terms of energy and financial cost in terms of data plans, if the devices are operating in GSM networks. In addition, the MDSM applications require persistent Internet connections in order to handle continuously streaming data. Considering these limitations, it is perceived that using far-edge mobile devices as a primary platform for application execution with heavy weight data processing can help in reducing dependencies over other far-edge mobile devices and cloud servers. The execution model must support MDSM applications in far-edge mobile devices, mobile edge servers, and CC servers.

The literature review also reveals that existing execution models mainly support static application execution. In addition, the dynamic and adaptive strategies mainly work at single far-edge device level. Existing literature still lacks the dynamic and adaptive execution models which support MDSM applications in multiple devices and systems especially in MECC settings. These dynamic and adaptive models need to enable ro-

bust application partitioning, computation offloading, data management, and data transfer strategies to fully support computationally complex MDSM applications.

2.5.1 Scope Limitation

Literature review shows that there exists countless opportunities in different dimensions to design and implement execution models for MDSM applications but the scope of this thesis is limited as follows.

- The aim of this research is to provide energy and memory efficient execution models for MDSM applications in MECC environment. However the models should be designed to minimize the dependency over cloud and Internet connections. In addition, maximum data should be processed in far-edge mobile devices to reduce the size of data stream which in turn minimize the cost of data communication.

- The focus of the research is on personal sensing devices to acquire and analyze the personal data stream. Because the applications based on personal analytics help in improving humane life in many aspects including life style, health, financial matters, leisure, and entertainment to name a few. Therefore, issues relevant to large-scale sensing and data processing systems were not considered in this thesis. These issues include large-scale collaborative data processing using multiple far-edge mobile devices, scalability, mobility, and parallel data stream offloading from multiple far-edge mobile devices.

- There exists many topological settings within MECC systems but the research focus is on device-centric and loosely coupled MECC communication model. The communication model works in three settings such as based on far-edge mobile devices, F2F communication where nearer peer far-edge mobile devices act as mobile edge servers, and F2C communication where a traditional CC system provides cloud

services for MDSM applications. The whole application execution is controlled by far-edge mobile devices.

- There are many kind of streaming data applications which could be deployed in MECC systems. However, the main scope of this is on personal analytics, IoT systems, and MECC systems. Therefore, all use-case applications in the experimental phase were designed relevant to mobile data stream mining operations. In addition, the proposed architecture provides the components for MDSM application execution and the dynamic and adaptive execution models provide the functionalities to cater the needs of MDSM applications.

2.6 Summary

The MDSM applications execute in multiple topological settings in multiple phases. Each phase of MDSM applications need to handle heterogeneity which increases the computational complexity. MDSM applications are deployed in different computing devices and systems with different form factors. This chapter reviewed different topological settings for MDSM platforms and execution models. In addition, a thorough review of execution models in terms of model type, model behavior, and granularity of data processing was presented. The chapter also presented a thorough review of supporting functionalities of execution models for data management, adaptive application execution, computation offloading, application partitioning, and data transfer among different devices and systems. Finally the chapter discussed the critical factors of complexity in MDSM applications and studied the perceived solutions for application deployment. In the next chapter, detailed feasibility and performance evaluation of far-edge devices is made to find the effect of critical factors of complexity. In addition, a framework for distributed MDSM applications in MECC systems is proposed and a three layer component-based architecture is proposed and formally verified using high level petri nets, SMT-Lib, and Z3 Solver.

CHAPTER 3: DISTRIBUTED DATA STREAM MINING IN MECC SYSTEMS

"...in many applications data stream can be read only once or a small number of times using limited computing and storage capabilities..."

ACM SIGMOD Record, Vol. 34, No. 2, (June 2005)

Chapter 2 presented the detailed literature review of topological settings and execution models. This chapter presents the problem analysis of heterogeneous mobile data stream mining (MDSM) applications in section 3.1. In section 3.2, a framework for distributed MDSM applications in MECC systems is proposed. In addition, a three-tier component-based architecture is proposed in this section. The formal verification and validation of proposed architecture is made in section 3.4. Section 3.5 presents the simulation results and discusses the perceived solutions for state explosion problem in proposed architecture. Finally, the chapter is summarized in section 3.6.

3.1 Problem Analysis

The resource limitations in mobile devices enforce the development of adaptive and light-weight data stream mining algorithms (Abdallah et al., 2015). Therefore, MDSM applications are bounded to compromise on the quality of knowledge patterns (Krishnaswamy et al., 2009). The problem analysis is performed to find the feasibility and performance of mobile devices as execution platforms for MDSM applications and uncover the resource consumption trends in terms of complexity factors as discussed in chapter 2.

In order to perform problem analysis, the experimental setup was configured in heterogeneous settings. Three resource constrained mobile devices, as specified in Table 3.1, were used. Since the selected devices offer different granularities of computational, memory, and battery resources therefore the experiments yielded more generalized results. The literature review in Appendix-A revealed that research on MDSM applications mainly

Table 3.1: Selected Devices.

	D1	**D2**	**D3**
Make	Samsung	Sony	Samsung
Model	GT-18552	Xperia Z3	Galaxy Tab S2
Operating System	Android 4.1.2	Android 5.1.1	Android 5.1.1
Processor	1.2 GHz Quad Core	2.5 GHz Quad Core	1.9 GHz Quad Core + 1.3 GHz Quad Core
Memory	1 GB	3 GB	3 GB
Battery Power	2000 mAh	3100 mAh	4000 mAh

covers algorithms for: (a) frequent pattern mining in order to find itemsets and association rules, (b) supervised classification to classify and predict the class variables, and (c) unsupervised and semi-supervised clustering to find the similarities/dissimilarities among data points in the data streams. The problem analysis is performed over nine data mining algorithms as presented in Table 3.2. These algorithms were selected because of their popularity and wide acceptance in existing literature (Agrawal & Srikant, 1994; Han, Pei, Yin, & Mao, 2004; Bache & Lichman, 2013; Hartigan & Wong, 1979; Fisher, 1987; Dempster, Laird, & Rubin, 1977; Quinlan, 2014; Lowd & Domingos, 2005).

Data stream mining algorithms work with variety of datasets, therefore, three different datasets were selected for each set of experiments. For frequent pattern mining algorithms (*i.e.* Apriori (Agrawal & Srikant, 1994), AprioriTid (Agrawal & Srikant, 1994), and FP-Growth (Han et al., 2004)), the retail dataset from UCI Repository was acquired (Bache & Lichman, 2013). Similarly, for clustering algorithms (*i.e.* *k*-means, Cobweb, and Expectation-Maximization(Hartigan & Wong, 1979; Fisher, 1987; Dempster et al., 1977)), weather dataset is downloaded from the same UCI repository. Finally, for classification algorithms (*i.e.* J48 (Quinlan, 2014), naive Bayes (Lowd & Domingos, 2005), and Random Forest (Breiman, 2001)), a mobile application was developed to collect the data streams from two mobile sensors that is accelerometer and GPS receiver. However, experimental settings were re-configured for each set of experiments in order

Table 3.2: Selected Algorithms.

No.	Algorithm	Objective
1	Apriori	Apriori algorithm is based on a tree data structure and performs knowledge discovery operations in order to find the itemsets and association rules among data points.
2	AprioriTid	AprioriTid algorithm improves Apriori by enabling multiple database scans in order to enhance the memory utilization. However the algorithm is based on multiple passes and it has iterative data processing behavior.
3	FP-Growth	FP-Growth algorithm handles the issue of increasing computational cost of candidate generation in large itemsets especially with long patterns. The algorithm is based on frequent-pattern (FP) tree to store important information about frequent patterns in compressed form.
4	J48	J48 algorithm uses decision-tree based data structures whereby nonlinear traversals are performed to classify the new instances.
5	Naive Bayes	Naive Bayes algorithm utilizes array-based data structures where the prior, conditional, and posterior probabilities of each class is calculated and the whole dataset is kept in memory for prediction.
6	Random Forest	Random Forest algorithm is based on multiple trees where each tree data structure maps a subspace from full dataset and separate nonlinear traversals are performed in each tree.
7	*k*-means	*k*-means algorithm is used to partition the data streams of n data points in the k clusters where n represents the whole size of datasets and k represents the number of required clusters. The cluster formation by *k*-means algorithm is done using the closest distance between cluster centroid, which is the mean value of all cluster data points, and the new observation.
8	Cobweb	Cobweb is a classification based hierarchical clustering algorithm that performs merge, split, insert, and object pass operations in the populated trees in order to cluster the data points.
9	Expectation-Maximization (EM)	The EM algorithm iteratively finds the maximum likelihood or maximum a posteriori estimates of parameters in statistical models. The EM models need unobserved latent variables in order to perform expectation and maximization operations on newly coming data points.

to assess the performance of mobile devices at different stages of application execution. The results in this section show the overall average resource consumption of MDSM algorithms.

3.1.1 Impact Analysis of Data Size and Data Rate

In order to perform impact analysis of data size, a continuous data stream was generated and given as input to data mining algorithms. The size of data stream was gradually incremented with 100 KB in each iteration whereas size of input data was increased from 100 KB to the maximum size of data as 20 MB. Since the mobile devices offer heterogeneous resources therefore the results of each algorithm on each device vary. However, the results show the average performance of mobile devices in terms of memory consumption, battery power consumption, and makespan which were measured in terms of Megabytes (MBs), milliwatts (mW), and milliseconds (ms), respectively.

For classification algorithms, as shown in Table 3.3, D1 outperformed in terms of memory consumption however D2 and D3 performed well in terms of battery power consumption and makespan. In addition, D1 processed data stream up to 7MB whereas D2 and D3 processed the whole input data streams. As D1 offered less memory as compared to D2 and D3 therefore memory availability by mobile operating system lowered which in turn affected the performance of MDSM applications. The comparison of results obtained from D1 with the results of D2 and D3 shows that although D1 consumed low memory, the average battery consumption and makespan increased between three to six times when compared with D2 and D3. Overall, the average memory consumption for classification algorithms in D1 remained 88.32 MB but it increased up to 221.28 MB and 160.10 MB in D2 and D3, respectively. The average battery power consumption in D1 was 167 mW but it decreased to 49 mW for D2 and 53 mW for D3. Similarly, the makespan of classification algorithms remained 48491 ms on average in D1 which lowered to 25468 ms in

Table 3.3: Performance Variation of Classification Algorithms with Respect to Data Size.

Data Size	D1			D2			D3		
	Memory	Power	Makespan	Memory	Power	Makespan	Memory	Power	Makespan
0.1 MB	84.35	100	2366	181.66	14	490	168.33	6	616
0.2 MB	84.66	96	4045	192.00	12	672	242.67	21	893
0.3 MB	85.33	108	6273	176.00	12	884	160.00	26	1206
0.4 MB	86.00	134	7543	178.00	25	1130	139.00	25	1385
0.5 MB	86.66	149	9882	180.33	22	1394	141.33	28	1720
0.6 MB	86.66	158	15377	181.66	21	1612	89.67	34	2014
0.7 MB	88.00	181	14944	184.67	20	1836	83.00	37	2326
0.8 MB	87.67	208	15766	186.00	21	2094	99.00	41	2554
0.9 MB	88.33	216	17852	188.67	23	2350	88.00	37	2839
1 MB	88.66	215	20418	190.33	25	2591	105.33	29	3205
2 MB	89.00	171	39339	194.76	59	4291	120.47	45	4729
3 MB	91.67	189	107780	201.77	62	6971	107.47	74	7659
4 MB	79.67	207	151123	205.93	69	9955	111.00	103	10453
5 MB	88.33	180	150240	218.33	51	13141	114.53	62	13440
6 MB	98.47	197	164415	218.57	48	16175	124.23	61	20185
7 MB	98.33	220	76791	200.27	46	19808	132.67	71	23851
8 MB	-	-	-	209.57	48	23108	133.43	42	22192
9 MB	-	-	-	211.10	49	25927	144.73	69	25073
10 MB	-	-	-	252.53	51	30205	149.83	49	28092
11 MB	-	-	-	251.37	88	33630	162.90	48	31090
12 MB	-	-	-	253.03	50	36372	166.13	70	34095
13 MB	-	-	-	218.43	59	40262	197.13	71	36946
14 MB	-	-	-	222.43	78	43848	214.00	52	39917
15 MB	-	-	-	236.03	55	46071	221.80	53	42805
16 MB	-	-	-	245.80	100	50231	220.90	61	47169
17 MB	-	-	-	250.36	140	52642	195.13	88	50406
18 MB	-	-	-	250.50	60	60461	213.93	106	53520
19 MB	-	-	-	281.87	51	63972	213.30	59	56678
20 MB	-	-	-	289.60	50	66915	225.93	60	60047

D2 and 24432 ms in D3.

Due to high computational complexity, the performance of clustering algorithm varied. It was observed that D1 was unable to run any of the clustering algorithms and D2 and D3 were able to process the data streams up to 12 MB. Therefore results presented in Table 3.4 are taken from D2 and D3. Although there is no significant difference between resource consumption on D2 and D3 but the devices consumed marginally higher battery and increased makespan when compared the results of classification algorithms. On average, D2 consumed 136.46 MB memory and 68 mW battery power with average makespan of 940347 ms. Alternatively, D3 consumed 139.82 MB memory and 44 mW battery power with average makespan of 1283248 ms. The results reveals that clustering algorithms are not a good choice for MDSM applications in far-edge mobile devices.

The results of frequent pattern mining algorithms, as presented in Table 3.5, show almost similar trends as classification algorithms. D1 was able to process the data stream

Table 3.4: Performance Variation of Clustering Algorithms with Respect to Data Size.

Data Size	D2			D3		
	Memory	Power	Makespan	Memory	Power	Makespan
0.1 MB	151.7	64	125088	153.7	27	138134
0.2 MB	143.0	50	301863	111.0	36	325934
0.3 MB	114.3	81	457009	118.3	40	562060
0.4 MB	86.00	48	631014	86.00	41	915923
0.5 MB	88.33	44	868036	86.33	40	1233786
0.6 MB	92.00	53	1006378	93.66	50	1515825
0.7 MB	100.0	125	1197814	101.7	44	1848598
0.8 MB	124.7	41	1452788	126.7	34	2653098
0.9 MB	109.7	44	1604983	110.0	37	1141158
1 MB	118.3	86	127458	145.3	45	1548428
2 MB	130.6	77	305661	134.6	61	471341
3 MB	117.0	88	463260	115.7	46	779034
4 MB	116.6	56	640116	117.9	50	1053057
5 MB	128.1	54	879522	125.4	53	1277688
6 MB	143.2	45	1020949	139.9	41	1558471
7 MB	174.4	133	1215678	174.7	53	2313476
8 MB	165.1	62	1474313	164.4	59	749175
9 MB	168.9	59	1628064	166.9	51	1101005
10 MB	201.1	124	1224627	181.7	41	1496484
11 MB	208.3	48	1483479	238.3	41	2024897
12 MB	184.3	49	1639192	245.3	43	2240633

up to 3 MB of data size however D2 and D3 processed the whole input data stream. The average memory consumption of D1 remained low as compared to D2 and D3 because of low memory availability by mobile operating system. Similarly, battery power consumption and makespan remained comparatively low in D2 and D3 because of high memory availability. On average, D1 consumed 49.25 MB of memory and 52 mW of battery power with makespan of 3375 ms. Alternatively, D2 consumed 131.58 MB memory, 114 mW battery power, and introduced makespan of 3540 ms. Similarly, D3 used 130 MB of memory, 33 mW of battery power, and 3794 ms in terms of makespan.

In addition with size of data streams, data rate impacts the performance of MDSM applications. The data rate is defined as the frequency of incoming data streams that MDSM applications need to handle. Since large size data streams increase makespan, waiting time for the next data streams increases. Therefore overall makespan increases for data streams with high data rates. It was observed that available memory size impacts

Table 3.5: Performance Variation of Frequent Pattern Mining Algorithms with respect to Data Size.

Data Size	D1			D2			D3		
	Memory	Power	Makespan	Memory	Power	Makespan	Memory	Power	Makespan
0.1 MB	53.33	32	701	154.3	16	124	188.7	10	106
0.2 MB	54.66	34	924	121.7	15	182	147.0	11	162
0.3 MB	55.33	37	1286	94.33	14	234	152.3	10	248
0.4 MB	55.33	39	1759	98.33	5	299	156.3	11	302
0.5 MB	56.33	46	2178	100.3	6	370	154.7	12	424
0.6 MB	57.66	56	2616	97.67	7	468	74.00	13	425
0.7 MB	58.33	61	2946	92.67	8	535	66.67	15	493
0.8 MB	59.66	66	3443	100.3	11	559	63.67	16	593
0.9 MB	34.33	53	3889	98.67	12	630	66.33	16	672
1 MB	34.33	50	4773	100.7	13	684	67.67	17	690
2 MB	36.46	69	7195	119.1	41	1074	75.17	25	1092
3 MB	38.50	81	8783	124.2	30	1757	85.50	33	1787
4 MB	-	-	-	136.0	36	2532	97.73	35	2561
5 MB	-	-	-	143.6	34	3285	110.5	37	6806
6 MB	-	-	-	122.5	603	3912	120.67	42	6207
7 MB	-	-	-	130.63	523	4875	130.43	80	5233
8 MB	-	-	-	140.3	66	5797	142.53	63	5837
9 MB	-	-	-	153.4	44	6694	155.8	46	6583
10 MB	-	-	-	163.3	415	7507	162.6	52	7639
11 MB	-	-	-	177.3	460	8338	178.8	47	8428
12 MB	-	-	-	186.7	122	9589	188.36	46	9233
13 MB	-	-	-	165.0	88	10395	199.63	62	10049
14 MB	-	-	-	205.7	45	11589	204.73	53	11698

the makespan. The comparison of results presented in Tables 3.3, 3.4, and 3.5, shows that the average makespan of data mining algorithms in D1 remained higher as compared to D2 and D3. Overall, the classification algorithms has lower makespan as compared to clustering and frequent pattern mining algorithms. It was also observed that clustering algorithms were not executed due to low memory availability in D1 and supported less size of data stream in D2 and D3 as compared to classification and frequent pattern mining algorithms. The performance of frequent pattern mining algorithms remained more or less similar to classification algorithms but in a few cases the frequent pattern mining algorithms consumed higher resources as compared to classification algorithms. The results show that data size and data rate significantly impact the performance of MDSM applications hence need to be controlled for efficient application execution.

3.1.2 Impact Analysis of Early Data Fusion and Data Preprocessing

In order to find and compare the impact of early data fusion and preprocessed data fusion approaches, the data streams were input in two different settings. In first setting, raw

data stream from accelerometer and GPS sensors was generated and sensor readings were concatenated to give as input to data stream mining algorithms. In second setting, the raw data stream was preprocessed using sliding windows and feature extraction based methods before inputting to data mining algorithms. The early and preprocessed data fusion methodologies are presented in Figure 3.1. The preprocessed data fusion approach works as follows. The mobile application collects 20 reading from accelerometer sensor every second and creates a new window after every 4 seconds. However windows data overlaps with previous window by 50% in order to minimize the data loss. The application uses four statistical methods (*i.e.* mean, median, standard deviation, and variance) over each window for feature extraction. The extracted features are appended with latest GPS location coordinates and the new feature vector is given as input to data stream mining algorithms. Table 3.6 presents the impact of early data fusion in D1, D2, and D3 whereas Table 3.7 shows the impact of late data fusion after data preprocessing. The statistics show that performance of each device varies with number of sensor readings wherein the devices with fewer resources (*e.g.* D1) consume comparatively low resources as compared to devices with higher resources such as D2 and D3.

The comparison shows that early data fusion increases memory consumption as compared to late data fusion. The memory consumption increased because MDSM applica-

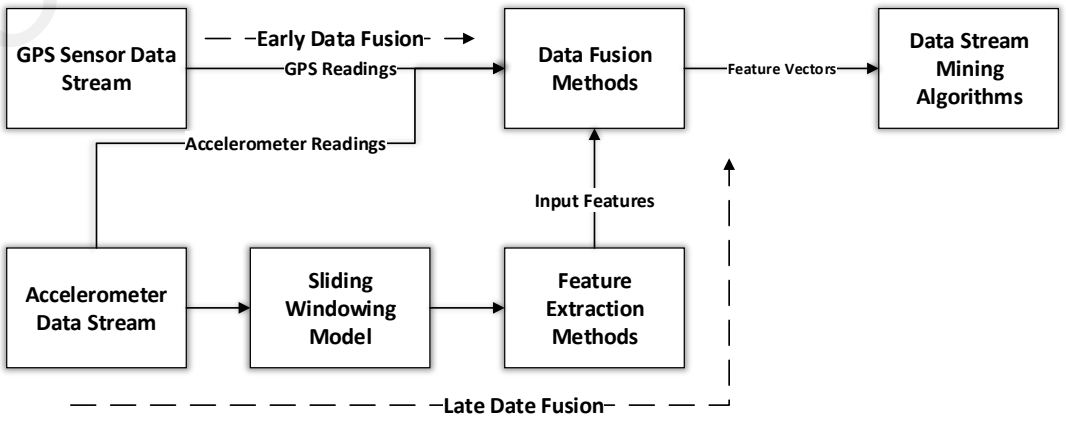


Figure 3.1: Early Data Fusion and Preprocessed Data Fusion Workflow.

Table 3.6: Performance Variation of Data Stream Mining Algorithms with Respect to Early Data Fusion.

No. of Readings	D1			D2			D3		
	Memory	Power	Makespan	Memory	Power	Makespan	Memory	Power	Makespan
2082	96.22	79	69	162.5	156	124	169.8	14	106
4164	93.88	71	58	152.2	132	182	166.7	23	162
6246	84.88	90	83	128.2	106	234	143.5	26	248
8328	75.77	87	62	120.7	100	299	127.1	26	302
10410	77.11	92	62	123.1	103	370	127.4	27	424
12492	79.00	100	70	127.7	104	468	85.77	33	425
14574	81.88	142	127	125.8	104	535	83.78	32	493
16656	90.67	117	70	137.1	113	559	94.44	30	593
18738	77.44	119	71	132.3	110	630	88.11	30	672
20820	80.55	137	103	136.4	113	683	106.1	30	690
41640	86.23	114	89	148.1	129	1074	110.1	44	1092
62460	78.38	127	101	147.6	129	1757	102.8	51	1787
83280	102.5	131	94	152.9	133	2532	108.9	63	2561
104100	113.3	117	85	163.3	143	3285	116.8	51	6806
124920	120.7	121	83	161.4	136	3912	128.2	48	6207
145740	134.2	176	92	164.4	144	4875	149.9	68	5233
166560	165.1	162	112	171.6	151	5797	146.7	55	5837
187380	168.1	159	109	177.82	162	6694	155.8	55	6582
208200	201.1	164	144	205.5	178	7507	164.7	47	7639
229020	208.3	148	98	212.3	189	8338	193.3	45	8428
249840	184.3	149	99	208.0	193	9590	199.9	53	9233
270660	-	-	-	191.73	172	10396	198.3	66	10049
291480	-	-	-	214.1	208	11588	209.3	52	11698
312300	-	-	-	236.0	221	46072	221.8	53	32805
333120	-	-	-	245.8	231	50232	220.9	61	37169
353940	-	-	-	250.4	235	52643	195.1	88	40406
374760	-	-	-	250.5	236	60462	213.9	106	43517
395580	-	-	-	281.9	267	63972	213.3	59	46678
416400	-	-	-	289.6	275	66915	225.9	60	50047
437220	-	-	-	289.1	274	64108	234.6	58	53481

tion need to store the data streams in memory buffer before sending for further processing. On the other hand, late data fusion strategies immediately process the data streams therefore lower the memory consumption. However the data preprocessing algorithms (such as feature extraction methods in this experiment) also use memory. Although in this experiment feature extraction methods used only 3 MB of memory at maximum to process any data window but computationally complex data preprocessing algorithms may increase the cost of memory consumption. In terms of memory consumption, late data fusion strategies are favorable in the case of data preprocessing algorithms with low computational complexities. However, a trade-off between early data fusion and the choice of data preprocessing algorithms is needed when the computational complexities of data preprocessing methods increase. The comparison of power consumption results shows that early data fusion consume more battery power (*i.e.* 213 mW on average) as compared to preprocessed data fusion (*i.e.* 67 mW on average). For example, the feature extraction

Table 3.7: Performance Variation of Data Stream Mining Algorithms with Respect to Preprocessed Data Fusion.

Number of Readings	D1			D2			D3		
	Memory	Power	Makespan	Memory	Power	Makespan	Memory	Power	Makespan
2082	53.00	64	71	151.7	155	294	152.6	6	46285
4164	54.00	50	60	121.7	130	389	110.7	11	108996
6246	55.00	81	84	94.33	103	494	118.3	10	187838
8328	55.33	48	68	86.00	96	624	86.00	11	305870
10410	56.33	44	76	88.33	99	765	86.33	12	411976
12492	57.00	53	77	92.00	101	895	74.00	13	506088
14574	58.00	121	128	92.67	101	1017	66.67	15	617139
16656	59.00	41	79	100.3	110	1142	63.67	16	885414
18738	34.00	44	81	98.67	107	1278	66.33	16	381556
20820	34.33	85	109	100.7	110	1405	67.67	17	517441
41649	36.46	77	93	119.1	127	2294	75.16	25	159054
62460	38.50	88	105	117.0	126	3712	85.50	33	262826
83280	88.33	56	94	116.6	129	5297	97.73	35	355357
104100	98.47	54	85	128.1	140	6967	110.5	37	432645
124920	98.33	45	83	122.5	133	8523	120.67	41	528287
145740	94.00	133	113	130.6	140	10460	130.43	53	780853
166560	105.7	62	112	140.3	148	12236	133.43	42	259068
187380	108.9	59	109	153.4	160	13790	144.73	46	377553
208200	121.1	124	144	163.0	174	15966	149.83	41	510737
229020	128.3	48	98	177.3	186	17768	162.9	41	688138
249840	108.3	49	99	184.3	191	19404	166.1	43	761320
270660	-	-	-	165.0	169	23625	197.1	62	23947
291480	-	-	-	205.7	207	27718	204.7	52	25807
312300	-	-	-	206.0	221	46072	211.8	53	42804
333120	-	-	-	215.8	221	60232	210.9	61	47169
353940	-	-	-	220.3	225	62643	185.1	88	50406
374760	-	-	-	220.5	226	70462	203.9	106	53516
395580	-	-	-	251.9	257	73972	203.3	58	56678
416400	-	-	-	259.6	265	76915	205.9	60	60047
437220	-	-	-	259.1	264	74108	214.6	58	63481

methods consumed 23 mW on average. Some computation-intensive data preprocessing operations, such as clustering based methods for outliers and anomaly detection, are not suitable for mobile devices which limits the scope of late data fusion methods.

The selection of data fusion methods impacts the makespan in MDSM applications. During early data fusion, raw data streams are aggregated from multiple data sources and the streams are immediately available for MDSM applications, therefore, the makespan remains minimal. Considering the successful executions upto 249840 readings in all three devices, average makespan during early data fusion in D1 was about 90 ms, in D2 the makespan increased upto 2830 ms, and in D3 the makespan increased upto 3120 ms. On the other hand, for late data fusion, the preprocessing operations increase makespan in MDSM applications. The average makespan in D1, D2, and D3 remained 93 ms, 5939 ms, and 432592 ms, respectively. It was observed that D1 maintained lowest makespan because the device was not able to process whole data stream due to low computational

and battery power resources. It could also be confirmed from existing literature (see Appendix-A) that preprocessed data streams are qualitatively better as it contain comparatively less noisy, anomalous, and missing data (Shoaib, Bosch, Incel, Scholten, & Havinga, 2014). In addition, the sizes of data streams are reduced that minimize the computational requirements at later stages in MDSM applications.

3.1.3 Impact Analysis of Learning Model Generation and Data Mining

As discussed in Chapter 2, the application components for learning model generation and performing data mining operations play critical roles in MDSM applications. The learning model generation is performed in training mode wherein the learning models are produced using supervised, unsupervised, and semi-supervised learning schemes. Therefore, the computational complexities of learnings algorithms vary significantly. On the other hand, data mining operations such as classification, clustering, and frequent pattern mining are performed in recognition mode wherein the data mining algorithms produce knowledge patterns using learning models. Earlier discussion in section 3.1.1 showed that clustering algorithms are not a feasible choice for mobile devices. On the other hand, frequent pattern mining algorithms implicitly learn from input data streams and do not require extra operations for learning model generation. The classification algorithms, however, require learning models separately in order to perform classification and prediction on input data streams. Considering these facts, the impact analysis of classification algorithms in training and recognition mode was performed.

The experimental setting was configured in two ways. First the learning models were trained in desktop PC environment wherein the labeled data streams were input into three WEKA (Hall et al., 2009) classifiers namely J48, naive Bayes, and random forest and the learning models were transferred in mobile devices to perform recognition. In the second setting, the learning models were trained onboard in mobile devices. To this

end, a separate mobile application was developed in order to label the data streams and train learning models. The training application performs data acquisition from onboard accelerometers, creates 50% overlapping sliding windows, and computes four statistical features (*i.e.* mean, standard deviation, variance, and magnitude) from each accelerometer axis values in the sliding window. The application allowed the users to annotate five physical activities (*i.e.* sitting, standing, walking, laying, and running) and the resultant feature vectors were given as input to train the learning models. The trained models were used by the same recognition application in mobile devices which was used for experiments in section 3.1.1. The evaluation is performed in terms of resource consumption (*i.e.* memory utilization, battery power consumption, and makespan) for learning model generation in mobile devices (see Table 3.8). In addition, the accuracy of classifiers in recognition mode is presented in confusion matrices for device-based (see Table 3.9) and desktop-based learning models (see Table 3.10).

On-device learning increases the computational burden and battery power consumption in mobile devices. Due to less memory availability in D1, learning model generation consumes comparatively more battery power and model development time also increases. However, D2 and D3 ensure comparatively more memory availability hence utilize less power and takes less time to develop learning models. The analysis of statistics presented in Table 3.8 shows that D1 utilized about 80% less memory as compared to D2 and D3 however it consumes about 58% extra battery power and model development time also increased about 76%. The comparison shows that on-device learning is a feasible choice as compared to desktop based learning as the developed model increases the accuracy level of data mining algorithms.

The problem analysis shows that mobile devices are good choice for MDSM applications. However, critical factors such as high volume and data rate of mobile data streams, the selection of early data fusion strategies, selection of preprocessing algorithms, learn-

Table 3.8: Impact of Learning Model Generation.

Data Size	D1			D2			D3		
	Memory	Power	Makespan	Memory	Power	Makespan	Memory	Power	Makespan
0.1 MB	24.35	68	3356	121.32	27	390	128.15	15	524
0.2 MB	24.94	56	3916	132.00	25	589	222.56	31	697
0.3 MB	26.23	108	5994	105.00	19	792	121.71	16	884
0.4 MB	27.03	78	6929	114.03	28	832	103.13	29	1483
0.5 MB	24.32	86	8389	120.05	29	1496	101.91	26	1430
0.6 MB	28.56	88	12184	121.54	19	1513	109.33	33	1915
0.7 MB	24.10	125	13764	121.86	25	1579	114.02	35	2014
0.8 MB	27.59	124	13422	121.65	27	1991	108.02	39	2192
0.9 MB	28.76	128	14232	122.39	28	2093	108.94	29	2934
1 MB	28.87	132	16402	130.48	28	2960	114.35	28	3557
2 MB	29.10	129	28039	138.26	29	3528	112.23	39	4985
3 MB	31.65	139	59359	141.65	32	3952	117.36	34	5808
4 MB	29.87	153	69035	135.34	43	4649	131.37	46	6479
5 MB	24.94	159	70333	144.35	51	5141	134.44	42	7435
6 MB	26.92	161	79393	145.61	48	6186	134.24	51	8298
7 MB	27.13	170	76757	146.36	53	7730	142.65	61	9454
8 MB	-	-	-	147.34	57	8108	143.44	63	10362
9 MB	-	-	-	149.23	67	8925	144.54	73	15072
10 MB	-	-	-	152.11	72	10530	145.77	71	18395

Table 3.9: Confusion Matrix for Device-based Learning Model.

Activities	Walking	Standing	Sitting	Laying	Running	Accuracy
Walking	**8821**	589	211	367	12	88%
Standing	1642	**8144**	112	93	9	81%
Sitting	584	246	**9138**	24	8	91%
Laying	891	10	138	**8827**	134	88%
Running	30	209	257	721	**8783**	88%
Overall Accuracy						**87.2%**

Table 3.10: Confusion Matrix for Desktop-based Learning Model.

Activities	Walking	Standing	Sitting	Laying	Running	Accuracy
Walking	**8092**	934	444	429	101	81%
Standing	1833	**7631**	294	151	91	76%
Sitting	893	357	**8617**	121	12	86%
Laying	923	516	122	**8243**	196	82%
Running	1358	414	271	58	**7899**	79%
Overall Accuracy						**80.2%**

ing model development and using on-device data mining algorithms, increase the memory and battery power consumption and makespan in MDSM applications. Therefore, it is not always feasible to use mobile devices as execution platform. Considering these limitations, a new MECC architecture is perceived wherein mobile devices could be used as primary platform for application execution. In case of resource scarcity, the execution support from other mobile devices or cloud data centers is acquired for efficient application execution in distributed settings.

3.2 UniMiner: A Framework for Heterogeneous Application Execution

Considering the problem analysis of MDSM applications in mobile devices and the need of a new MECC architecture, this section presents a novel reference framework for execution of distributed MDSM applications in MECC systems. The framework, named as UniMiner, enables three-tier execution model in MECC systems using far-edge mobile devices, mobile edge servers, and CC servers. Figure 3.2 presents the detailed execution workflow of MDSM applications using UniMiner. The framework enables multiple operations at application level for data acquisition and adaptation, data preprocessing, and data fusion. It facilitates platform level operations for transient data stream management, and data stream offloading based on resource monitoring, context collection, and size of unprocessed data. The UniMiner enables mobile-based data analytics in standalone and collaborative settings and cloud based data analytics for remote data processing. In addition, UniMiner ensures knowledge availability in mobile devices and cloud platforms.

At the application level, the data acquisition operations are performed to collect the data streams from onboard and off-board, and, sensory and non-sensory data sources. In addition, the data adaptation operations enable to configure the data size and data rates in MDSM applications in order to handle computational complexity at lateral stages of application execution. The data preprocessing operations help in improving the quality of acquired data streams by enabling multiple preprocessing operations that are used for feature extraction, anomaly detection, outliers detection, handling inconsistencies, and conversion of data stream from unstructured data to semi-structured and fully structured data formats. UniMiner ensures data fusion strategies in order to collect and synthesize the data streams from multiple data sources in MDSM applications.

At the platform level, UniMiner perform transient data stream management operations in order to enable robustness in MDSM applications by avoiding data loss and

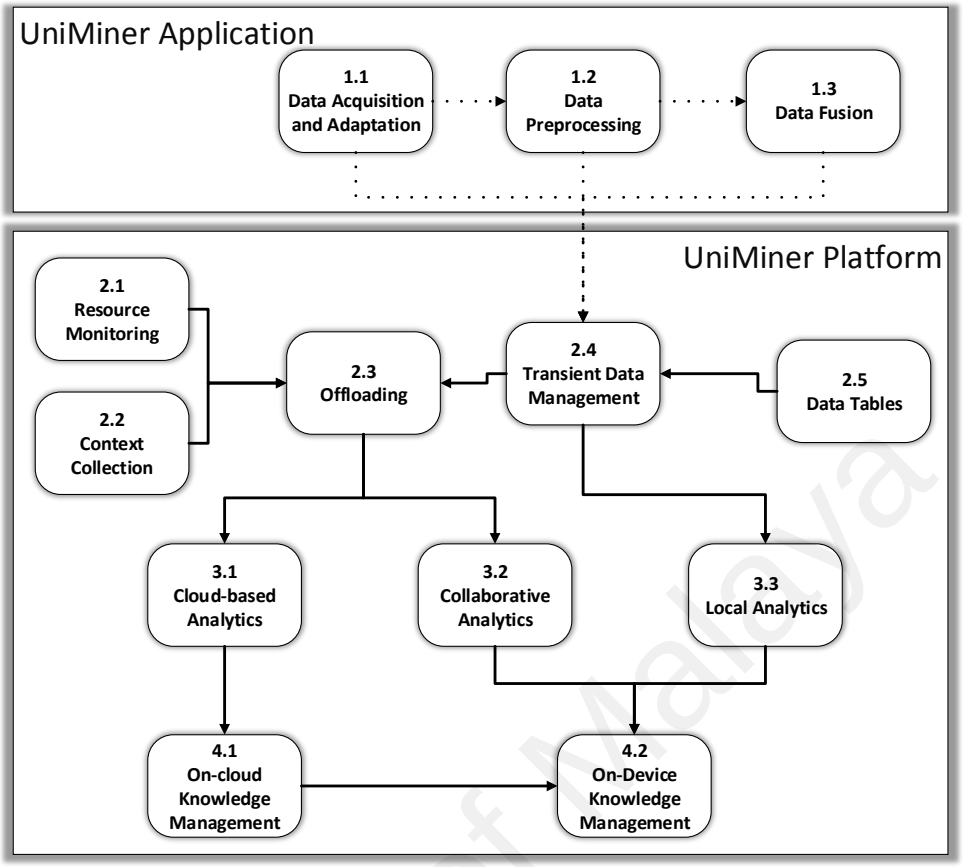


Figure 3.2: UniMiner Framework for Distributed MDSM Application Execution in MECC Systems.

storing data in transient data stores. The transient data stream management schemes are enabled using data tables that stores the traces of stream executions for efficient data management. In case of resource scarcity in far-edge mobile devices, UniMiner enables data stream offloading in mobile edge servers and cloud data centers. The offloading decision is performed on the basis of resource monitoring information and contextual information collected by UniMiner applications. In addition the amount of unprocessed data in transient data stores is considered for offloading decisions.

UniMiner enables three layers of application execution. The local analytics layer facilitates in onboard execution of application components. Conversely, the collaborative analytics layer functions as a group of far-edge mobile devices in the same communication area that execute MDSM applications collaboratively. The cloud enabled analytics

layer facilitates with cloud-based service model wherein the cloud services are used to augment mobile devices with unlimited computational power and storage space. The UniMiner facilitates in knowledge availability in mobile and cloud environments. The knowledge patterns are synchronized among mobile devices and cloud data centers by ensuring local and cloud based knowledge availability.

3.2.1 Assumptions

Although UniMiner supports generic application execution; following assumptions are made in this thesis.

- The framework is designed to enable device-centric MDSM applications whereby the decision of processing in far-edge mobile devices, mobile edge servers, and CC servers is performed solely in far-edge mobile devices.
- The mobile device can function as a client or a server but not both at the same time.
- The collaboration between client mobile devices and mobile edge servers is performed only in the case wherein the mobile edge servers offer more computational resources and battery power as compared to offloading client mobile device.

3.3 Three-layer MECC Architecture for UniMiner

This section proposes a component based MECC architecture for UniMiner. The UniMiner enables three layers of execution for MDSM applications. (see Figure 3.3). These three layers include local analytics layer (LA) for onboard execution, collaborative analytics layer (CA) for execution of MDSM applications in mobile ad-hoc network, and cloud-enabled analytics layer (CLA) for the provision of data stream mining services in cloud computing systems. The architecture follows the component based development (CBD) approach for future enhancements and interoperability with other mobile devices (wearable devices, WSNs, BSNs, and Laptops) and cloud computing systems.

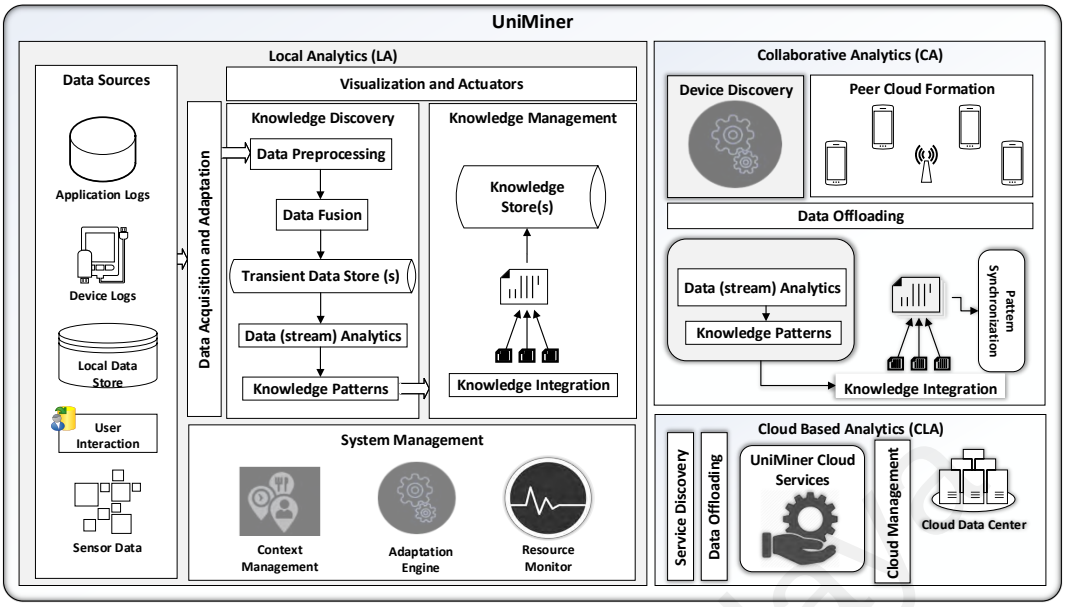


Figure 3.3: UniMiner's Component-based Architecture.

3.3.1 Components and Operations for LA

At LA layer, the UniMiner provides five modules for *a*) data acquisition and data adaptation, *b*) knowledge discovery, *c*) knowledge management, *d*) visualization and actuation, and *e*) system management.

3.3.1.1 Data Acquisition and Data Adaptation

The UniMiner applications start execution and run as backend services in mobile devices. Primarily, these applications perform the intelligent data collection depending upon application requirements. The data collection strategy also varies in different applications. For example, some of the applications (like environmental monitoring applications) may collect continuous data streams and some of the applications may collect situation-based or periodic data collection. The data acquisition and adaptation module provides functionalities to execute various data collection strategies according to application requirements.

3.3.1.2 Knowledge Discovery

The knowledge discovery module supports execution of analytic components using onboard computational resources in mobile devices. The module provides different components for data preprocessing operations such as noise reduction, outliers detections, handling missing values, and anomaly detection to name a few. In addition, the data fusion components provide functionalities to aggregate data streams from multiple homogeneous and heterogeneous data sources such as onboard and off-board sensors, and Internet enabled social media data streams. Moreover, the module provides components for transient storage of fused data streams in mobile devices. Furthermore, the module provides a library of different data stream mining algorithms in order to perform clustering, classification, and association rule mining operations.

3.3.1.3 Knowledge Management

The knowledge patterns generated by knowledge discovery module differ depending upon the selected algorithms. The knowledge management module enables to integrate the relevant knowledge patterns and produce a summarized and global view of overall information. The knowledge integration is made in a way that all processed data could be effectively represented and resultant data is stored in local data stores using light databases. However, the challenge of tracking the processed and unprocessed data points introduces the complexity in overall data management process. To address this issue, the UniMiner works in the principles of data parallelism where each chunk of raw data is tracked from data acquisition to integration of knowledge patterns. The data parallelization ensures that each data chunk is processed at least one time.

3.3.1.4 System Management

The onboard resource dynamics and fast mobility create the issues of tracking device locations and onboard available resources. The core components of system management

module are adaptation engine, context monitor, and resource monitor. These components ensure the robustness of UniMiner architecture in different scenarios. The adaptation engine ensures the execution of UniMiner components for MDSM applications in all three layers. In addition, the resource monitor and context manager periodically monitor available resources, locations (frequently visited locations), and device-usage behavior (charging, idol) in order to support adaptation engine for seamless application execution.

3.3.1.5 Visualization and Actuation

The visualization module ensures the local knowledge availability by enabling on-screen visualization. The studies show that local visualization is very beneficial for real-time applications. However, due to resource-constraints and limited screen size, local knowledge visualization do not support detailed knowledge view. The topic of visualization needs a detailed and thorough study therefore it is not covered further in this thesis. The actuation component is designed to ensure the interaction of mobile devices with external environments which include remote cloud services and nearby peer devices. This module ensures the future extensibility of UniMiner to other devices and systems.

3.3.2 Components and Operations for CA

The discovery of mobile edge servers and the available communication interfaces are key requirements for ad-hoc cloud formation using far-edge mobile devices. The execution of knowledge discovery processes collaboratively and synchronizing resultant knowledge patterns is challenging in mobile ad-hoc cloud. The CA layer of UniMiner handles these issues to ensure seamless and collaborative application execution. The components at CA layer enable connectivity, communication, data processing, and synchronization operations.

3.3.2.1 Discovering Mobile Edge Servers and Communication Interfaces

The device discovery modules handle two main issues. First, it discovers the far-edge mobile devices that may function as mobile edge servers. The source device in the network scans all connected communication interfaces and enlists all available mobile devices. The adaptation engine in UniMiner maintains and periodically updates a list of mobile devices in order to use them as mobile edge servers. The known devices are given priority over unknown devices. However, the list of known devices is maintained and updated whenever a new device is connected. This approach helps to seamlessly adopt in new and unknown environments for collaborative application execution. The second main issue handled by UniMiner at this stage is to adapt and switch between different communication interfaces. It ensures to seamlessly switch between different communications interfaces while maintaining the proximity of devices. This strategy helps to ensure maximum collaboration considering co-movement between different communication areas (*i.e.* Wi-Fi networks, public Internet facilities, and home-networks).

3.3.2.2 Peer to Peer (P2P) Network Formation

Once the mobile edge servers are found and the information about their communication interfaces is collected, the UniMiner initiates P2P network formation process. The source device broadcasts the peering request to all proximal mobile edge servers, which collect the information about available onboard computational resources and send them back to source device. The source device then performs the cost-benefit analysis in order to decide the favorability of data offloading. In case of favorable data offloading, far-edge mobile device offloads data stream to connected mobile edge servers.

3.3.2.3 Knowledge Discovery and Pattern Synchronization

Once offloading is completed, mobile edge servers execute the components using knowledge discovery modules. However it depends upon the application design whether the

whole knowledge discovery process is executed at the mobile edge server or partial task execution is performed. In case of complete execution, far-edge device offloads raw data streams and mobile edge server executes complete knowledge discovery process from preprocessing to data mining and summarization of patterns. In case of partial execution far-edge device offloads only preprocessed data in order to lower the overall bandwidth utilization in ad-hoc network. However, mobile edge server executes rest of the knowledge discovery process and resultant patterns are synchronized with far-edge device. In case, far-edge device could not receive the results from mobile edge server for a specified time period, the data streams are offloaded to any other available mobile edge server. To lessen the transient storage burden and to reserve the maximum computational power, the garbage collection process is executed by UniMiner and mobile edge servers delete all processed raw data streams periodically from RAM and device's local storage. Similarly, far-edge mobile device performs garbage collection periodically.

3.3.3 Components and Operations for CLA

The CLA layer represents the clone of mobile applications augmented with feature rich data mining and knowledge management services. The key issues in CLA are the service availability and orchestration of cloud services for knowledge discovery and knowledge management.

3.3.3.1 Service Discovery and Service Model

The UniMiner maintains a service repository of available cloud services. The requirement of cloud services varies therefore service repository contains various services for remote data reduction in cloud computing systems. The choice of service is solely dependent upon the needs of MDSM applications however the UniMiner provides interface to access all available services in the repository. The MDSM application offloads data stream in cloud environment with request for required cloud services where cloud service

manager automatically runs the requested services and completes the task execution. The UniMiner provides seven types of services which are designed for data uploading, data preprocessing, data fusion, data mining, pattern summarization, knowledge management, and pattern synchronization. The data uploading services help in handling offloaded data streams. The raw data streams are uploaded in transient data stores in cloud computing systems. The data preprocessing, data fusion and data mining services are executed in order to process raw data streams and uncover new knowledge patterns. The pattern summarization and knowledge management services are used to integrate and summarize knowledge patterns from both uploaded by mobile devices and produced by cloud services. The summarized knowledge patterns are stored in permanent data stores inside cloud computing systems. The pattern synchronization services transfer the knowledge patterns for data aggregation in cloud data centers for future usage of knowledge patterns.

3.4 Formal Modeling, Analysis and Verification

The UniMiner architecture is formally modeled, analyzed, and verified using high level petri nets (HLPN), satisfiability modulo theories library (SMT-Lib), and Z3 solver. The basic introduction to HLPN, SMT-Lib, and Z3 solver is provided by researchers in (Diaz, 2013; De Moura & Bjørner, 2009, 2008) to aid readers' understanding therefore further discussion on the topic is not made in this thesis.

3.4.1 High Level Petri Nets

Petri Nets are used for graphical and mathematical modeling of a system and are applied to a wide range of systems, such as distributed, parallel, concurrent, nondeterministic, stochastic, and asynchronous. For the formal modeling of UniMiner, we used a variant of conventional petri net called HLPN. The HLPN simulates a system and provides its mathematical properties that are used to analyze the behavior of a system. HLPN is based on 7-tuple model $N = (P, T, F, \varphi, R, L, M_0)$, where P denotes a set of places, T refers to the

set of transitions (such that $P \cap T = \emptyset$), F denotes flow relation (such that $F \subseteq (P \times T) \cup (T \times P)$), φ maps places P to data types, R denotes a set of rules for transitions, and L is a label on F and M_0, which represents the initial marking. (P,T,F) provides information about the structure of the net and (φ,R,L) provides the static semantics (information) that does not change throughout the system. In HLPN, places can have tokens of multiple types, which can be a cross product of two or more types. A few mapping examples include $\varphi(P1) = Boolean$, $\varphi(P2) = ID$, $\varphi(P3) = P(Integer)$, and $\varphi(P1) = Char$, where P1, P2, and P3 are the places of HLPN.

3.4.2 SMT-Lib and Z3 Solver

SMT is used for verifying the satisfiability of formulae over theories under consideration. SMT-Lib provides a common input platform and benchmarking framework that helps in the evaluation of the systems. The usage of SMT is common in many fields including deductive software verification. This thesis adopts Z3 solver with SMT-Lib that is a theorem prover developed at Microsoft Research. Z3 is an automated satisfiability checker that determines whether the set of formulas are satisfiable in the built-in theories of SMT-Lib. HLPN model for UniMiner framework is shown in Figure 3.4. We identify data types, places, and mapping of data types to places. Data types and their mappings are shown in Tables 3.11 and 3.12, respectively. In Figure 3.4, the rectangular black boxes represent transitions and belong to set T, and circles represent places and belong to set P.

For each data collection phase, the *Data_Sources* information is initialized and the data collection is started. The time series buffered data stream is created using *T_Stamp*, temporary file name (*F_Name*) and system generated data sources' ID (*DS_ID*). UniMiner generates all unique IDs in the system at the time of application deployment. Therefore, these IDs remain constant until the application is installed on a device. When the collected data files reach a maximum threshold (*i.e.* file size given by application developer), the

Table 3.11: Data Types for UniMiner HLPN.

Types	**Descriptions**
T_Stamp	A DateTime type representing date and time
F_Name	A string type representing unprocessed data file name
DS_ID	A string type representing name of data source
Chunk_ID	A string type representing data chunk code generated by system
Flag_Status	An integer type representing status of data chunks (unprocessed, processed, processing)
Location	A string type representing the name and GPS coordinates of a location
Charging	A Boolean type representing the charging status of mobile device
Locked	A Boolean type representing the lock status of mobile device
Calling	A Boolean type representing the call status of mobile device
Internet	A Boolean type representing the availability status of active Internet interfaces
Dev_ID	A string type representing the device_id based on IMEI of mobile
Mem	An integer type representing maximum memory in mobile device
Storage	An integer type representing maximum storage in mobile device
App_ID	A string type representing application_id in the mobile device
Avlb_Loc_Storage	An integer type representing available local storage in mobile device
Avlb_SD_Card	An integer type representing available storage on SD-card in mobile device
Wi-Fi	A string type representing availability and connectivity status through Wi-Fi
GSM	A string type representing availability and connectivity status through GSM
BT	A string type representing availability and connectivity status through Bluetooth
BL	A string type representing availability and connectivity status through Blue Tooth Low Energy
Exec_Mode	A string type representing current execution mode of the system
Pattern_Attributes	A string type representing multiple attributes of extracted patterns (type of patterns, number of patterns, quality of patterns)

Table 3.12: Places and Mappings.

Places	**Mappings**
ϕ(Data_Sources)	ρ(T_Stamp $\times$ F_Name $\times$ DS_ID)
ϕ(Loc_Data)	ρ(T_Stamp $\times$ F_Name $\times$ DS_ID)
ϕ(Data_Tab)	ρ(Chunk_ID $\times$ Flag_Status $\times$ DS_ID $\times$ F_Name)
ϕ(Up_Data)	ρ(Chunk_ID $\times$ Flag_Status $\times$ DS_ID)
ϕ(Context_Info)	ρ(T_Stamp $\times$ Location $\times$ Charging $\times$ Calling $\times$ Internet $\times$ Locked)
ϕ(Conn)	ρ(Dev_ID $\times$ Mem $\times$ Storage)
ϕ(Local_Res)	ρ(T_Stamp $\times$ Mem $\times$ Avlb_Loc_Storage $\times$ Avlb_SD_Card $\times$ Wifi $\times$ GSM $\times$ BT $\times$ BL)
ϕ(Est_Res)	ρ(App_ID $\times$ Mem $\times$ Storage)
ϕ(Exec_Mode)	ρ(Exec_Mode)
ϕ(Disc_Pattern)	ρ(Chunk_ID $\times$ Pattern_Attributes)
ϕ(Intr_Pattern)	ρ(Chunk_ID $\times$ Pattern_Attributes)
ϕ(Loc_Pattern)	ρ(Chunk_ID $\times$ Pattern_Attributes)
ϕ(Cloud_Str)	ρ(Chunk_ID $\times$ DS_ID $\times$ Pattern_Attributes)

data file is stored on the onboard storage. In addition, a *Flag_Status* is maintained for each data file (called data chunk). The *Flag_Status* shows the current processing status of any data chunk (*i.e.* 0 for unprocessed, 1 for under-processing, and -1 for processed). This is done at transition $T1$ and the transition is mapped to the following rule (see Equa-

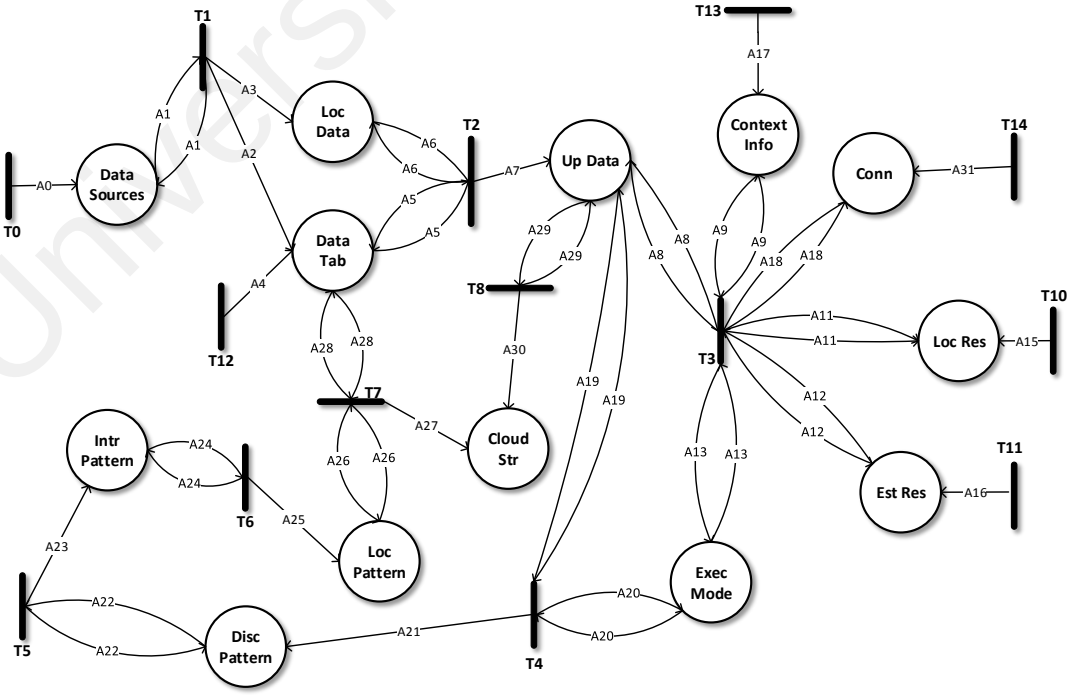


Figure 3.4: UniMiner HLPN Model.

tion 3.1).

Once the sufficient data is collected, $T2$ gathers data from *Loc_Data* and corresponding attributes (*Chunk_ID*, *DS_ID*, and $Flag\_Status$) from *Data_Tab* and updates $Flag\_Status$ to "under-processed" (*i.e.*, 1). In addition, $T1$ periodically cleans processed data from *Loc_Data* and updates *Data_Tab* accordingly. The data controlling rule at $T2$ is mapped as follows (see Equation 3.2).

$$\begin{aligned} R(T1) = \forall a1 \in A1 \bullet a1[1] \neq NULL \wedge a1[2] \neq NULL \wedge a1[3] \neq NULL \wedge \\ \forall a3 \in A3 \bullet a3[1] := a1[1] \wedge a3[2] := a1[2] \wedge a3[3] := a1[3] \wedge \\ \forall a2 \in A2 \bullet \exists a2[1] := dist(a1[2]) \wedge a2[2] := a1[3] \wedge a2[4] := a1[2] \wedge \\ \forall a4 \in A4 \wedge A3' = A3 \cup (a3[1], a3[2], a3[3]) \wedge A2' = A2 \cup (a2[1], a2[2], a2[3], a2[4]) \wedge \end{aligned} \tag{3.1}$$

$$\begin{aligned} R(T2) = \forall a5 \in A5, \forall a6 \in A6 \bullet a5[4] = a6[2] \wedge a5[3] = 0 \wedge \\ \forall a7 \in A7 \bullet a7[1] := a5[1] \wedge a7[2] := a5[3] \wedge a7[3] := a5[2] \wedge \\ A7' = A7 \cup (a7[1], a7[2], a7[3]) \wedge a5[3] := 1 \wedge A5' = A5 \cup (a5[1], a5[2], a5[3], a5[4]) \wedge \end{aligned} \tag{3.2}$$

After the establishment of amount and type of data to be processed, $T3$ collects $Context\_Info$, Conn, *Loc_Res*, *Est_Res* related information and executes *Rule_Engine* which runs execution rules and switches between all three execution modes (*i.e.*, LA, CA, or CLA). The rule for selection of execution mode is mapped at $T3$ (see Equation 3.3).

$$R(T3) = \forall a8 \in A8, \forall a9 \in A9, \forall a10 \in A10, \forall a11 \in A11, \forall a12 \in A12, \forall a13 \in A13,$$
$$\bullet a13[1] = LA - Mode \wedge a12[2] > a11[2] \wedge \forall a12[3] > a11[3] + a11[4] \wedge$$
$$a13 := CA - Mode \wedge \forall a18 \in A18 \mid a10[1] = a12[1] \wedge$$
$$a12[2] > a18[3] \wedge a12[3] > a18[4] \wedge a13 := CLA - ModeA13' = A13 \cup (a13) \tag{3.3}$$

$T4$ collects $Exec\_Mode$ status and in case of LA and CA, data mining tasks are initiated locally. For LA, all data mining tasks are executed using onboard local resources. However, in case of CA, data mining tasks are offloaded to mobile edge servers, where each mobile edge server acts as a standalone data mining platform. In addition, $T4$ collects Up_Data and schedules the data mining tasks accordingly. The $Exec\_Mode$ at $T4$ is mapped using the following rule (see Equation 3.4).

$$R(T4) = \forall a19 \in A19, \forall a20 \in A20, \forall a21 \in A21 \bullet a21[1] := a19[1] \wedge$$
$$a21[2] := Data - Mining(a19[1]) \wedge A21' = A21 \cup (a21[1], a21[2]) \tag{3.4}$$

Once the data mining tasks are executed successfully, the $Disc\_Pattern$ are evaluated at $T5$ and is mapped as follows (see Equation 3.5).

$$R(T5) = \forall a22 \in A22, \forall a23 \in A23 \bullet a23[1] := a22[1] \wedge a23[2] := a22[2] \wedge$$
$$A23' = A23 \cup (a23[1], a23[2]) \tag{3.5}$$

After refinement of $Disc\_Patterns$ into $Intr\_Patterns$, the relevant patterns are summarized and merged at T6 using Equation 3.6.

$$R(T6) = \forall a24 \in A24, \forall a25 \in A25 \bullet a25[1] := a24[1] \wedge a25[2] := a24[2] \wedge A25' = A25 \cup (a25[1], a25[2]) \tag{3.6}$$

$T7$ synchronizes *Loc_Patterns* with cloud data stores (*Cloud_Str*). In addition, the $Flag\_Status$ of successfully executed *Chunk_ID* is updated to "processed" (*i.e.*, -1). The synchronization at $T7$ is mapped using Equation 3.7. Consequently, whenever the Internet connection is available and neither of LA, CA, or CLA is enabled and there are some unsynchronized data patterns (UPs), then UPs are synchronized with their respective counterparts in the cloud. However, data patterns need to be stored in different directories on far-edge mobile devices to classify synchronized and unsynchronized versions.

$$R(T7) = \forall a26 \in A26, \forall a27 \in A27, \forall a28 \in A28 \bullet a27[2] := a26[2] \wedge a28[1] = a26[1] \wedge a28[3] = -1 \wedge a27[1] := a26[1] \wedge A27' = A27 \cup (a27[1], a27[2], a27[3]) \wedge A28' = A28 \cup (a28[1], a28[2], a28[3], a28[4]) \tag{3.7}$$

Lastly, in case of *CLA*, raw data stream is uploaded to *CloudStr* using following rule (see Equation 3.8).

$$R(T8) = \forall a29 \in A29, \forall a30 \in A30 \bullet a30[1] := a29[1] \wedge A30' = A30 \cup (a30[1], a30[2], a30[3]) \tag{3.8}$$

3.5 Simulation Results and Discussion

The formal verification of HLPN (using Z3 solver) determines that UniMiner is completely workable and executes according to specified properties. UniMiner was also eval-

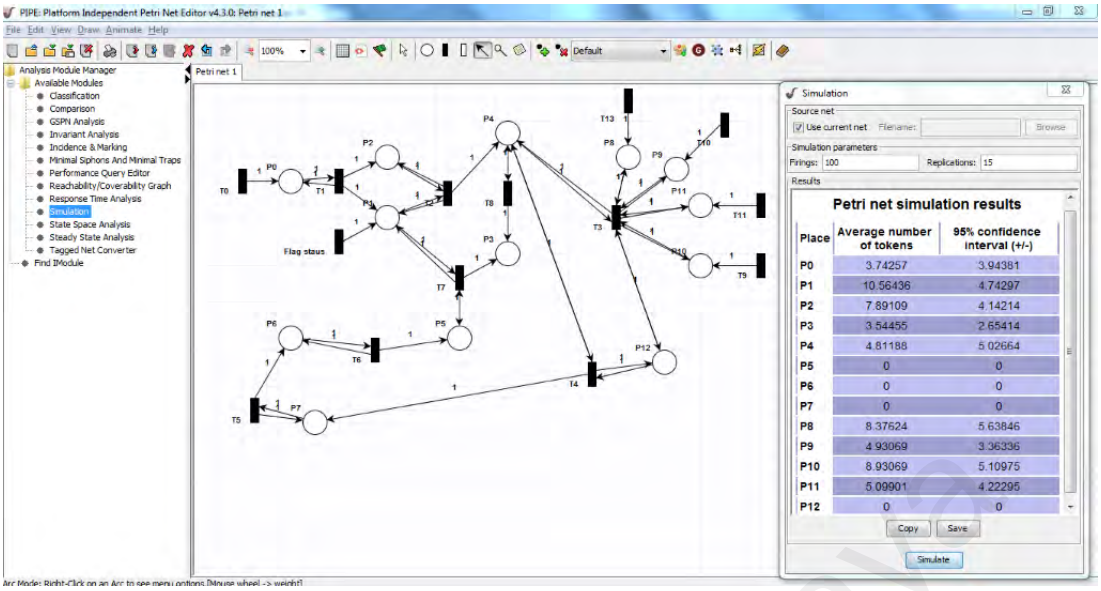


Figure 3.5: PIPE+ Editor Screenshot of UniMiner.

uated using PiPE+ editor (Bonet, Lladó, Puijaner, & Knottenbelt, 2007) which provides graphical interface to develop and analyze HLPN for bounded model checking (BMC) (see Figure 3.5). The traversal paths in HLPN are given in forward and backward incidence matrices generated using PiPE+ (see Tables 3.13, 3.14). The results show that all places in the UniMiner are reachable when moving forward (see Table 3.13). Similarly, all places, except φ(Cloud_Str), are reachable in reverse order (see Table 3.14). The φ(Cloud_Str) is made irreversible to eliminate the loop in the data processing cycle.

BMC handles state space explosion problem by executing limited number of states. Therefore, BMC is applied over finite set of transitions (M) using a linear temporal logic

Table 3.13: Forward Incidence Matrix.

Places	**T0**	**T1**	**T10**	**T11**	**T6**	**T5**	**T4**	**T12**	**T2**	**T7**	**T8**	**T3**	**T13**	**T14**	**T19**
ϕ(Data_Sources)	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
ϕ(Loc_Data)	0	**1**	0	0	0	0	0	0	**1**	0	0	0	0	0	0
ϕ(Exec_Mode)	0	0	0	0	0	0	**1**	0	0	0	0	**1**	0	0	0
ϕ(Loc_Pattern)	0	0	0	0	**1**	0	0	0	0	**1**	0	0	0	0	0
ϕ(Intr_Pattern)	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0
ϕ(Disc_Pattern)	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0
ϕ(Data_Tab)	0	**1**	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0
ϕ(Up_Data)	0	0	0	0	0	0	**1**	0	**1**	0	**1**	**1**	0	0	0
ϕ(Cloud_Str)	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0
ϕ(Context_Info)	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0
ϕ(Conn.)	0	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0
ϕ(Loc_Res)	0	0	**1**	0	0	0	0	0	0	0	0	**1**	0	0	0
ϕ(Est_Res)	0	0	0	**1**	0	0	0	0	0	0	0	**1**	0	0	0

Table 3.14: Backward Incidence Matrix.

Places	**T0**	**T1**	**T10**	**T11**	**T6**	**T5**	**T4**	**T12**	**T2**	**T7**	**T8**	**T3**	**T13**	**T14**	**T19**
ϕ(Data_Sources)	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
ϕ(Loc_Data)	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0
ϕ(Exec_Mode)	0	0	0	0	0	0	**1**	0	0	0	0	**1**	0	0	0
ϕ(Loc_Pattern)	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
ϕ(Intr_Pattern)	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
ϕ(Disc_Pattern)	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
ϕ(Data_Tab)	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0
ϕ(Up_Data)	0	0	0	0	0	0	**1**	0	0	0	**1**	**1**	0	0	0
ϕ(Cloud_Str)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ϕ(Context_Info)	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
ϕ(Conn.)	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
ϕ(Loc_Res)	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
ϕ(Est_Res)	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0

(LTL) formula (f) and given upper bound value 'k'. BMC determines an execution path of length 'k' that satisfies LTL formula. For BMC, first of all a logic formula φk is constructed from M, f, and k and verified using constraint solver. If an 'f' is satisfied over a path of maximum length 'k' in M_k then φk is said to be satisfiable. In existential BMC, it is very hard to find the upper bound for 'k',therefore, negated safety property is used for validation. The negation of safety property determines the safety of the mode as long as f is not satisfiable. The HLPN model was translated to logic formulas (as discussed in subsection 3.4.2) and evaluated its satisfiability. The tokens are distributed in different places in various markings on each state of HLPN.

In general, during safety (reachability) analysis PiPE+ generated 3072 states and 38592 arcs creating a space explosion problem when $\varphi(Data\_Sources)$ and $\varphi(Exec\_Mode)$ are enabled with one token on each place. The simulation results (see Table 3.15) shows that all places are reachable and satisfies the safety property. The minimum thresholds, where all places are reachable are 36 firings with 5 replications. Alternately, maximum threshold is 10000 firings with 15 replications. The simulation results with minimum and maximum thresholds are presented in terms of average number of tokens produced at each place and acceptable margin of error during each execution cycle. Consequently, the argument is established that proposed UniMiner architecture is completely workable and all places are reachable using specified rules.

Table 3.15: Simulation Results of UniMiner HLPN model.

Places	Minimum Threshold		Maximum Threshold	
	Avg. No. of Tokens	**95% Conf.**	**Avg. No. of Tokens**	**95% Conf.**
ϕ(Data_Sources)	3.5135	1.7770	334.8233	18.1603
ϕ(Loc_Data)	2.9189	0.6936	354.3783	27.6568
ϕ(Exec_Mode)	4.2703	0.6006	684.8284	38.9546
ϕ(Loc_Pattern)	2.1892	1.2426	342.7440	20.0734
ϕ(Intr_Pattern)	1.72973	1.15794	675.6431	41.8029
ϕ(Disc_Pattern)	2.4054	1.1550	340.9338	20.8442
ϕ(Data_Tab)	0.4595	2.18681	323.2102	31.7452
ϕ(Up_Data)	1.0270	1.4271	332.9073	32.9875
ϕ(Cloud_Str)	1.5405	0.4962	305.2618	28.0542
ϕ(Context_Info)	0.9729	1.0296	320.1321	30.2847
ϕ(Conn.)	1.0270	0	1.0001	0
ϕ(Loc_Res)	0.5405	0.7352	339.9139	31.5361
ϕ(Est_Res)	0.3514	0.5164	327.8112	25.2031

Considering thees results, UniMiner needs to address state-explosion problem in order to control the application execution process. The state explosion problem is addressed by proposing data stream management and opportunistic data stream offloading schemes.

3.6 Summary

Chapter 3 presented problem analysis to adapt mobile devices as execution platform for MDSM applications. However, experimental evaluation revealed that it is not always feasible to utilize far-edge mobile devices as standalone execution platform for MDSM applications. Considering this limitation, a novel framework for distributed MDSM applications was proposed in this chapter. The framework supports MDSM applications at three levels. Primarily, far-edge mobile devices are used as execution platform but the framework supports application execution using mobile edge servers and CC servers. The proposed framework was formally modeled and analyzed using HLPN and Z3 solvers and simulated through PiPE+ editor. The simulation results show that UniMiner needs to address the state explosion problem for seamless application execution in MECC systems. Chapter 4 presents the solutions to address state explosion problem using dynamic and adaptive execution models for MDSM applications in MECC systems.

CHAPTER 4: EXECUTION MODELS FOR MDSM APPLICATIONS IN MECC SYSTEMS

"...distributed system is a model in which components on networked computers communicate and coordinate their actions by passing messages..."

Coulouris, George; Jean Dollimore; Tim Kindberg; Gordon Blair (2011)

Chapter 3 presented a thorough discussion on performance analysis of mobile data stream mining (MDSM) applications when executed strictly in far-edge mobile devices. In addition a three-tier computing architecture was proposed for distributed MDSM applications. The proposed three-tier architecture ensures the provision of computing resources near far-edge devices in order to minimize the makespan and bandwidth utilization, and reduces the dependency over persistent Internet connections. However, the formal analysis shows that the proposed architecture has the state explosion problem wherein the MDSM application can enter in any state of execution without centralized control. Chapter 4 presents the problem of data-intensive and compute-intensive MDSM applications and presents two execution models in order to handle state explosion problem for seamless application execution in mobile edge cloud computing (MECC) systems.

4.1 Preliminaries

This section presents the discussion on MDSM application architecture. In addition, it presents the preliminary discussions on MDSM application's state transition model, and discusses the data-intensive and compute intensive MDSM applications.

4.1.1 MDSM Application Architecture in MECC System

The MDSM applications are based on different application components and their mutual interactions. The MDSM application and its components are defined as follows.

- ***Definition 1:*** An MDSM application in MECC system is defined as a program or

a set of programs that perform predefined analytic operations using MECC architecture. The MDSM application in MECC system maintains the distributed logic among far-edge mobile devices, mobile edge servers, and CC systems.

- ***Definition 2:*** An MDSM application consists of one or several components where each component performs specific operations. The distribution of application components solely depends upon the application model and required functionality of MDSM applications.

4.1.2 Multistage MDSM Application Execution in MECC Systems

Considering the complexity and heterogeneity in MECC systems, the MDSM application execution process is redefined in this thesis. The MDSM applications are based on multistage execution process (see Figure 4.1) where the nature of data stream and analytics operations change at every stage. This multi-stage execution process is defined as follows.

- ***S1: Data Stream Acquisition,*** The data stream acquisition components provide functionality to acquire data streams from one or more data sources. Formally, data stream, DS, is an infinite set of data tuples, Td_i, continuously collected by MDSM applications (see equation 4.1). Depending upon the number of data sources and nature of MDSM application, the number of attributes in each Td_i varies. However the DS contains raw data points and requires further processing before learning model generation and performing data mining operations.

$$DS = \begin{pmatrix} Td_1 \\ Td_2 \\ Td_3 \\ \vdots \end{pmatrix} \tag{4.1}$$

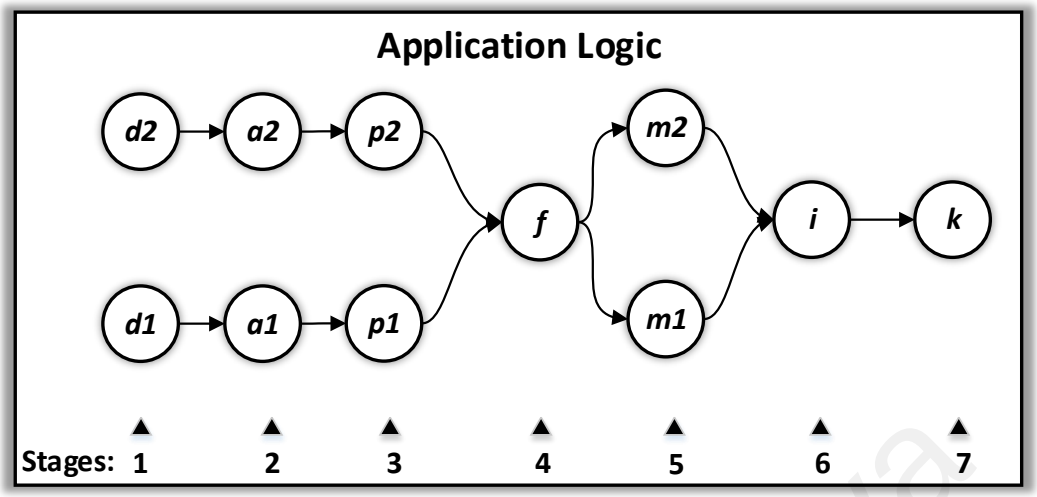


Figure 4.1: Multistage Application Execution

Property 1: The number of data sources and nature of each MDSM application may vary therefore the number of attributes in each Td_i varies accordingly.

Example 1: An MDSM application that help in activity detection collects data streams from onboard accelerometer in Smartphone. The application collect three data points *i.e.* x-axis, y-axis, and z-axis for each reading. However, the MDSM application which provides additional features of users' locations collects five data points *i.e.* x-axis, y-axis, z-axis, latitude, and magnitude.

- ***S2: Data Rate Adaptation,*** The data rate in each MDSM application vary according to the functionality. Therefore, the functions performed by data rate adaptation components vary accordingly.

Example 1: The data rate adaptation strategy for location based activity detection vary when the MDSM application detect indoor activities such as sleeping, sitting, and standing as compared with outdoor activities such as running and driving. Since the indoor locations are not supposed to be changed quickly therefore low data rate is more adequate as compared to high data rate. Conversely, high data rate in outdoor locations can help in detecting users' activities at numerous places.

- ***S3: Data Preprocessing,*** The data preprocessing components enable to improve the quality of data stream by handling missing values, removing noise, and detecting anomalies and outliers. Formally, the selection of preprocessing algorithms varies according to four properties of *DS*.

  Property 1: The data type of attributes, A_b, varies therefore the selection of preprocessing operations vary accordingly.

  Example 1: The A_b in transactional data streams of e-commerce applications have 'integer' and 'floating-point' data types for product code and price, respectively (Rehman et al., 2014a). Similarly, the data types of A_b in sensor data streams of mobile activity detection applications are 'floating-point' and 'date-time' (Shoaib et al., 2014). In e-commerce applications the data preprocessing components mainly find missing values and detect outliers and anomalous transactions however in mobile activity detection it perform preprocessing using feature extraction and noise reduction methods.

  Property 2: The nature of A_b in *DS* is mainly dependent upon the data rate of incoming data streams. Therefore MDSM applications perform different kinds of preprocessing operations.

  Example 2: The data stream applications need to perform sketching and sampling operations in order to handle high data rates (*i.e.* number of T_d per second) (Cormode & Muthukrishnan, 2004; Cormode, Garofalakis, Haas, & Jermaine, 2012). On the other hand, the applications with low data rate perform in-memory preprocessing operations (Gama, 2013). Similarly, the MDSM applications which involve constant data rate adopt different preprocessing operations as compared with data streams having variable and uncertain data rates (Chen & Chen, 2014).

Property 3: The sparsity of attributes A_b in *DS* varies according to the number of A_b in the *DS* therefore the selection of preprocessing operations vary accordingly.

Example 3: The number of A_b in *DS* may vary from single attribute to hundreds or thousands of attributes. The preprocessing operations for a few attributes may require statical features extraction based methods however for highly sparse data streams the MDSM applications use complex dimension reduction algorithms (Amini, Wah, & Saboohi, 2014; Guan, Wang, Duan, & Ji, 2015).

Property 4: The density of attributes A_b in *DS* varies according to the number of missing values therefore the selection of preprocessing operations vary accordingly.

Example 4: The missing values of A_b in *DS* are represented by the number of un-populated instances of each T_d. Therefore the preprocessing components to handle density in *DS* varies accordingly (Cameron, Cuzzocrea, Jiang, & Leung, 2013).

- ***S4: Data Stream Fusion,*** The fusion of data streams from multiple data sources results in information rich data stream representing multiple facets of each data tuple. Formally, a fused data stream, D, contains time-stamp information, TS_a, and, b, number of attributes, A_b, derived from each data source. The data sources in mobile environments produce multi-format information; therefore, all preprocessed information is concatenated using data fusion methods represented by join operator $\bowtie$ (see equation 4.1).

$$D = \begin{pmatrix} Td_1 \\ Td_2 \\ \vdots \\ Td_a \end{pmatrix} = \bowtie \begin{pmatrix} TS_1 & A_{1,1} & A_{1,2} & \cdots & A_{1,b} \\ TS_2 & A_{2,1} & A_{2,2} & \cdots & A_{2,b} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ TS_a & A_{a,1} & A_{a,2} & \cdots & A_{a,b} \end{pmatrix} \tag{4.2}$$

The complexity in D increases when Td_a also incorporates semi-structured data (*e.g.* Tweets, and Facebook posts) and unstructured data (*e.g.* device log files and audio/video recordings using onboard cameras) from multiple data sources.

- ***S5: Data Mining,*** The data stream mining operations are performed for online and offline knowledge discovery using different model-based and model-less data mining algorithms. Formally, the online data stream mining algorithms are defined as the in-memory operations (Cameron et al., 2013). The online algorithms collect the data stream and perform run-time operations using available onboard memory and computational resources and discard the data streams immediately after processing. The online algorithms usually work as one-pass data mining algorithms whereby no iterative data processing is allowed. Alternatively, the offline data mining algorithms perform data collection and data processing operations in batch mode. On the contrary, the offline algorithms are not memory-bounded therefore it involve iterative data processing (Larose, 2014).

- ***S6: Knowledge Integration,*** The uncovered knowledge patterns are summarized and integrated for further utilization using knowledge integration components (Shi, Zhang, Tian, & Li, 2015). Formally, the knowledge patterns such as clusters, predicted classes, itemsets, and association rules are summarized for a holistic view of whole knowledge discovery processes. For example, at each iteration, the clustering algorithms reshape the cluster structures; the classification algorithms update the summary of predictions, and the association rule mining algorithms update the association rules among different itemsets. The summarized knowledge patterns are integrated and utilized by MDSM applications. The knowledge integration involves homogeneous (*i.e.* within the same class of algorithms) and heterogeneous (*i.e.*with different classes of algorithms) approaches.

- ***S7: Knowledge Management,*** The knowledge management components store and provide integrated knowledge patterns for further utilization by MDSM applications. Formally, knowledge management components provide functionality to store knowledge patterns in mobile devices, mobile edge servers, and CC servers. However, the knowledge management strategies differ in MDSM applications. For example, the MDSM applications that deal with personal data and facilitate individual users may store knowledge patterns using onboard storage. This strategy ensures makespan-minimal knowledge availability and reduced dependency over Internet connections. Alternatively, MDSM applications that perform knowledge discovery operation on *DS* acquired from multiple users may use mobile edge servers and cloud computing (CC) servers as knowledge stores.

4.1.3 MDMS Application State Transition Model

The maintenance of execution status of different program components define the execution states. The MDSM application components preserve one state at a particular instance of time (see Figure 4.2). The state is either "Null", "Running", "Paused", "Terminated", or "Contextualized". The application states are changed by "pause", "resume", and "stop" commands invoked after some predefined operations such as input data collection, task completion by a program component, resource availability in far-edge devices, and changes in contextual information.

Normally, during multistage application execution, the state transition model maintains "Null", "Running", "Paused", and "Terminated" states. The "Contextualized" state is invoked to find the adequate execution mode depending upon the resource availability and resource requirements, availability of nearer far-edge mobile devices as edge servers, and availability of Internet connections to communicate with CC servers. The system management components for resource monitoring, context collection, and adaptation en-

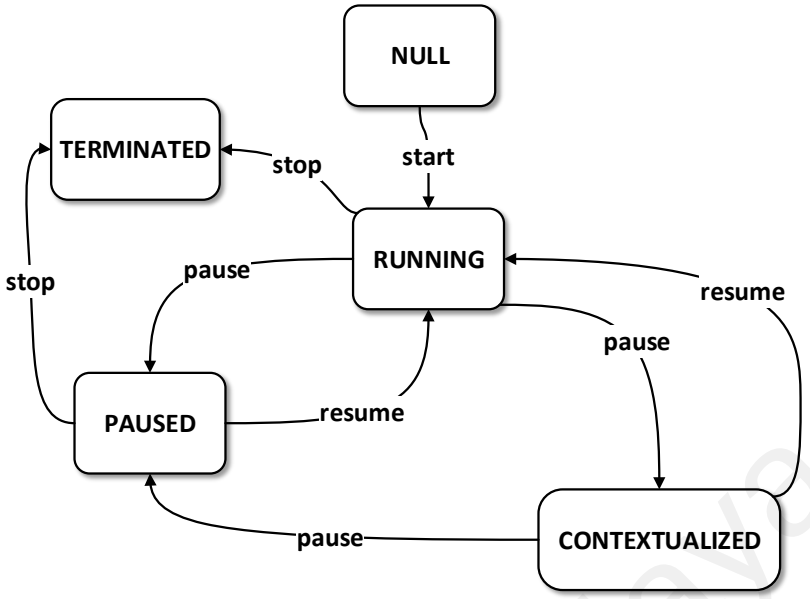


Figure 4.2: State Transition Model of MDSM Applications.

gine run only when the MDSM application enters in "Contextualized" state. In addition, the configuration of components vary in different MDSM application execution models. The resource monitoring and contextual information is collected using onboard application components named as *resource profiler* and *context profiler*. The *adaptation engine* enables to execute MDSM applications seamlessly in MECC systems.

1. **Resource profiling:** The *resource profiler* component enables to periodically monitor and profile onboard computational and battery power resources in far-edge mobile devices. The *resource profiler* provides information about memory, CPU, and energy utilization of currently executing MECC application. In addition, *resource profiler* monitors the currently available resources in mobile device. The information provided by *resource profiler* is further utilized by *adaptation engine* to perform the cost-benefit analysis and assess the feasibility of offloading decisions.
2. **Context collection:** Mobile devices move in different communication areas therefore the communication interfaces frequently change among Wi-Fi and GSM Internet connections. In addition, mobile users have different device usage behaviors

such as timings and places of charging far-edge mobile devices, calling and device usage, running specific applications at specific times such as mobile activity detection application during exercises and ambulatory activities, and blood glucose monitoring after breakfast. Similarly, other contextual information can help in efficient application execution. For example, maintaining a list of frequently available far-edge mobile devices in user proximity, or Wi-Fi routers in home, office, Gym, and other frequently visited locations may help in predicting the availability of mobile edge servers that could be opportunistically utilized, when required. The *context profiler* component enables to maintain such contextual informations which are further utilized by *adaptation engine* for lateral executions.

3. **Adaptation Engine:** The *adaptation engine* performs cost-benefit analysis over device capability and current resources. It also enables rule-based scheduling strategies to execute MDSM applications using dynamic and adaptive execution models.

4.1.4 Data-intensive vs. Compute-intensive Applications

In MECC systems, MDSM applications address multiple issues relevant to, 1) application and platform level heterogeneity, 2) critical factors of complexity, 3) variations in data-intensities, and 4) computational complexities of application components. Considering the above mentioned issues, MDSM applications are categorized into four groups as shown in Figure 4.3. As confirmed by the empirical results in Chapter 3, the MDSM applications with low data-intensity and low computational complexity seamlessly execute in far-edge mobile devices. Alternatively, high data-intensity applications with high computational complexities run in edge cloud computing systems. However, other MDSM applications need to be dynamically and adaptively executed in MECC systems.

In order to execute high data-intensity applications having low computational complexities, the dynamic execution model is proposed in this thesis. The model is perceived

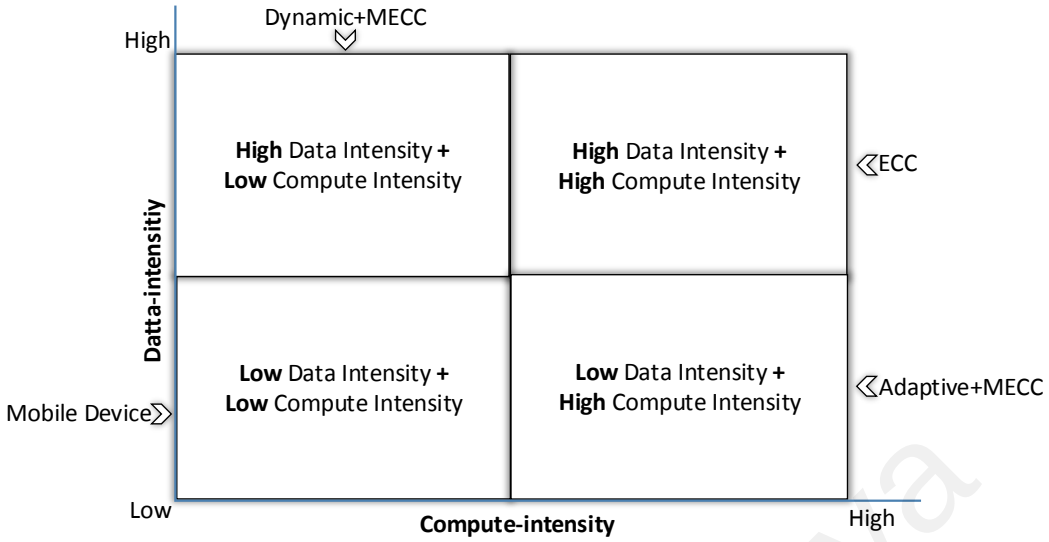


Figure 4.3: Data-intensive vs. Compute-intensive MDSM Applications in MECC systems.

to be dynamic because it enables the opportunistic execution of MDSM applications in MECC systems whereby MDSM applications dynamically switch among far-edge devices, mobile edge servers, and CC servers. Alternatively, the adaptive execution model is proposed to execute MDSM applications with low data-intensity but high compute-intensity. The model is said to be adaptive because the MDSM applications switch the execution behavior from multiple points whereby application components at each point have different compute-intensities. Due to variable computational complexities, MDSM applications on far-edge mobile devices with low computational capabilities and minimal battery power switch to mobile edge servers or CC servers at early stages of executions. On the other hand, the MDSM applications with high computational resources and sufficient battery power execute maximum components using onboard resources and switch to mobile edge servers or CC servers at later stages of application execution. This strategy helps far-edge devices to adapt the execution behavior according to underlying resources.

4.2 Dynamic Execution Model

The main objective of dynamic execution model is to perform maximum data processing in mobile environments when MDSM application components have low computational complexity and do not require unbounded memory, CPU, and battery power resources.

4.2.1 Operations

As witnessed in Chapter 3, low intensity MDSM applications work with low data rate and minimal data size therefore the data preprocessing, data fusion, learning model generation, and data mining operations do not significantly impact the performance of far-edge mobile devices. The dynamic execution model partially executes MDSM applications in far-edge mobile devices and rest of the application components either run in far-edge mobile devices, mobile edge servers, or CC servers (see Figure 4.4). The data acquisition, data preprocessing, data adaptation, and data fusion components of MDSM applications are executed strictly inside far-edge mobile devices. The applications generate intermediate data files after performing data fusion operations and temporarily save in onboard local storage. In order to manage the data files, the execution model enables transient data stream management scheme. In addition, the model uses an opportunistic offloading scheme to decide whether the remaining application components for data mining, knowledge integration, and knowledge management should be executed in far-edge mobile devices, mobile edge servers, or CC servers.

The dynamic execution model has following constraints. The application components for data acquisition, data adaptation, data preprocessing, and data fusion must not be computationally complex. In addition, mobile edge servers must have more computational (CPU, Memory) and current battery power as compared to offloading far-edge mobile device. Finally, the collaborating far-edge mobile devices and mobile edge servers must be in the same communication area.

4.2.2 Single-point Data Stream Management

After performing data fusion operations in far-edge mobile devices, the model facilitates transient data stream management. The scheme facilitates in temporarily storing data streams so that the proposed opportunistic offloading scheme can help in finding adequate execution environment such as far-edge mobile device, mobile edge servers, or CC servers. Since the execution model enables device-centric data processing wherein far-edge mobile device controls the application execution process therefore data streams collected by single far-edge mobile device do not quickly hampers onboard storage resources. Hence transient data stream management is performed separately at each far-edge mobile device.

Definition 1: F_i is a data file containing 'i' number of data tuples, where $i > 0$, and F_i is bounded by maximum number of data tuples set by the application designer.

Example 1: From definition 1, F_i is the data file that contains preprocessed data after performing data fusion operations, for example for an activity detection application, F_i contains feature vectors that are obtained after performing statistical feature extraction methods and fusing the resultant features values with GPS coordinates. The application designer sets the maximum number of data tuples that can be stored in any F_i. However, F_i is randomly generated to dynamically utilize the onboard mobile device resources.

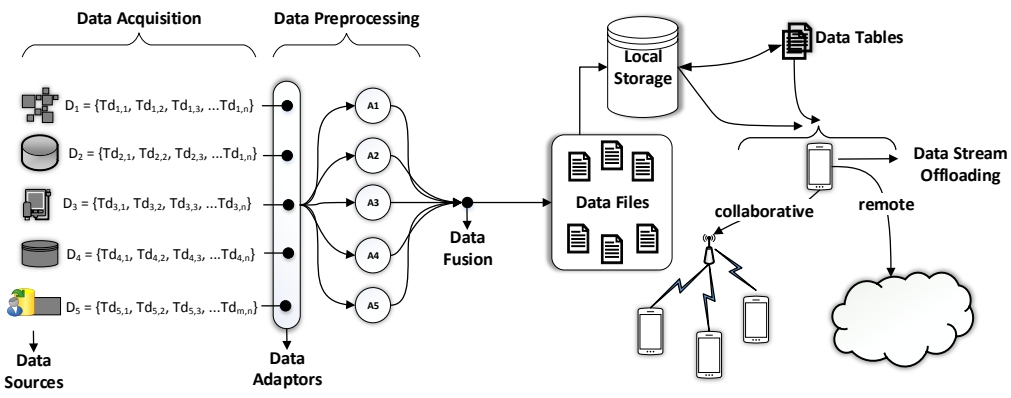


Figure 4.4: Dynamic Application Execution Model.

4.2.2.1 Managing Data Tables

The application identifier, A_{id}, that is derived from process id of running application, data store identifier, DS_{id}, which is derived from application manifest, and time stamp TS_a are used to generate file identifier, f_{id}, for each F_i. The preprocessed data files are stored in onboard local data stores with a system generated f_{id} . In addition, the system maintains a data table entry with two attributes for f_{id} and status, $S_{f_{id}}$,. A hash function, $h(f_{id})$, is used to access F_{id} whenever required and updates the corresponding ($S_{f_{id}}$) with 0 for unprocessed, 1 for under-processing, and -1 for processed F_i. Once the F_i are successfully processed the $h(f_{id})$ function updates the data tables and deletes corresponding files from transient data stores using $u(f_i)$ function.

4.2.3 Single-point Opportunistic Data Stream Offloading

Data stream offloading in MECC systems becomes complex when considering multiple objectives like energy efficiency, performance enhancement, data reduction, and bandwidth utilization cost. In addition, the offloading decision becomes more complex due to mobile devices joining/leaving different networks, limited resource constraints and dynamically changing available communication interfaces when users move among different communication areas. In view of the data stream offloading complexity, a rule-based scheduling strategy is hereby proposed to switch between far-edge mobile devices, mobile edge servers, and CC servers. The scheduling process functions on the basis of contextual information (*i.e.* current location, communication interfaces and available nearer edge servers), available resources (*i.e.* memory, storage and battery), unprocessed data, and estimated resource consumption (*i.e.* memory, battery and CPU) of MDSM application required for the current data. Once the input parameters are established, the rule-based scheduling is invoked for opportunistic data stream offloading. The successful execution of a rule defines the execution mode and the environment for offloading is enabled.

4.2.3.1 The Proposed Offloading Strategy

The data stream offloading strategy, as presented in Algorithm 1, is invoked whenever an unprocessed F_i is created by MDSM applications and saved in transient data stores. The offloading algorithm collects contextual information about the device usage (P_{idle}, P_{lock}), and charging status D_{ch} (see line 2). Due to personal nature of data streams and the requirement for efficient utilization of onboard computational resources, the algorithm halts the execution if device is being used for calling, or user is interacting with the device. However, execution resumes if the user is interacting with MDSM application UA_i, the nature of UA_i is real-time, or the device is locked or being charged. Once execution starts, the offloading algorithm calculates the file size S_{F_i} and estimates the resource-consumption (see line 7) of CPU cycles C_r, memory M_r, and battery B_r for lateral processing of data mining algorithms A_{dm}. The acquired information is compared with available resources (C_a, M_a, B_a) and offloading algorithm sets the execution environment for data processing.

The MDSM application switches to mobile edge server if onboard resources are insufficient. The algorithm scans all communication interfaces CI_n (see line 23) and returns list of connected mobile edge servers (CD_n). Familiar mobile edge servers (CD_f) are ranked on the basis of familiarity, and information is collected about available resources (C_{cd}, M_{cd}, B_{cd}) at CD_n . Cost-benefit analysis is performed on the far-edge mobile device (see line 26-27) and in case of feasible offloading, mobile device and mobile edge server exchange information about communication interfaces (EC_{mw}, EC_{bt}, EC_{ble}). Far-edge device offloads F_i at each active interface and most energy efficient interfaces are paired for lateral communication (see line 29-32). Meanwhile, algorithm monitors the execution by setting a timer. If results are not received before the timer expires, F_i is rescheduled. Finally, the results are synchronized and F_i are deleted from mobile edge server.

Algorithm 1: Opportunistic Data Stream Offloading

Global: $List\_res\_consumed[100]$
Input: F_i
1. **for all** F_i repeat 2 to 45
2. $CollectContext()$, $getCurrentPower()$
3. **if** $(!P_{lock}$ && $!P_{idle}$ && $UA_i.status \neq real-time)$ || $(P_{idle}$ && $!D_{ch}$ && $CP \leq 10\%)$
4. $WaitandCollectData()$
5. **else**
6. $SF_i = SizeOf(F_i)$
7. $estimateResources(SF_i, List\_res\_consumed[])$
8. $calculateAvailableResources()$
9. $consumed\_resources[4] = null$
10. **if** $(C_a > T_{avg}[].C_r$ && $M_a > T_{avg}[].M_r$ && $B_a > T_{avg}[].B_r$ && $SF_i < T_{avg}.FileSize)$
11. execute $local_{full}$
12. $consumed\_res[4] = SF_i, B_{used}, C_{used}, M_{used}$
13. update($List\_res\_consumed[100]$, $consumed\_res[]$)
14. **else if** $(C_a > T_{max}[].C_r$ && $M_a > T_{max}[].M_r$ && $B_a > T_{max}[].B_r$ && $SF_i < T_{max}.FileSize)$
15. execute $local_{full}$
16. $consumed\_res[4] = SF_i, B_{used}, C_{used}, M_{used}$
17. update($List\_res\_consumed[100]$, $consumed\_res[]$)
18. **else if** $(UA_i == `realtime')$
19. execute $local_{adaptive}$
20. $consumed\_res[4] = SF_i, B_{used}, C_{used}, M_{used}$
21. update($List\_res\_consumed[100]$, $consumed\_res[]$)
22. **else**
23. scan $(CI_n[])$
24. $FindandRank(CD_n, CD_f)$
25. $CollectResourceInformation(CD_n[])$;
26. **if** $((F_i.count > 0)$ && $(C_{cd} > C_r$ && $M_{cd} > M_r$ && $B_{cd} > B_r$
27. && $(CD_{idle} || CD_{ch} || CD_{lock} || CD_{UA_i})))$
28. $Scan(EC_{mw}, EC_{bt}, EC_{ble}, CD_n)$
29. **for all** CI_n
30. Offload (F_i, CI_n)
31. EC[] = calcEnergyConsump(CI_n)
32. SortandPrioritize(EC[])
33. Time = $T_{avg}[].Time$;
34. **while** $(((F_i.count > 0)$ && $(C_{cd} > C_r$ && $M_{cd} > M_r$ && $B_{cd} > B_r$ &&
35. $(CD_{idle} || CD_{ch} || CD_{lock} || CD_{UA_i} == True))))$
36. StartTimer(Time)
37. Send (F_i, CI_n)
38. **if**$(time > 0)$
39. Synchronize(P_i)
40. Garbage(F_i)
41. **else if** $(WiFi \neq null || GSM \neq null)$
42. prioritize(CI_{wf}, CI_{gsm})
43. offload (F_i, A_i)
44. DiscoverandRunServices (F_i, A_i)
45. Synchronize($P_i, localStorage, CloudDatabase$)

The MDSM application switches the execution to CC servers in case of resource unavailability in mobile device (C_a, M_a, B_a), and mobile edge servers (C_{cd}, M_{cd}, B_{cd}). The algorithm offloads F_i and A_{dm} in CC server where relevant services are discovered by cloud resource manager and the knowledge discovery process is completed. Finally, the results (P_x) are synchronized between far-edge mobile device and CC data stores.

The proposed data stream offloading strategy ensures complete process execution in MECC environments. However, data stream offloading is disabled when a real-time application is working in resource-critical (*e.g.* battery $\leq$10%) situations and there is no mobile edge server or Internet-enabled communication interface. In this situation, the algorithm facilitates to work in low processing mode ($local_{adaptive}$) (see line 18-21) by disabling some resource-sucking communication interfaces (*e.g.* GSM data connection) and onboard sensors. Alternately, the applications with tolerable makespan only work in data collection mode to acquire maximum data before complete battery charge depletion. This strategy helps to reduce data loss and the data stream offloading is enabled whenever the MDSM application is activated after device restarts.

4.2.3.2 Dynamic Resource Estimation for Data Stream Offloading

The resource estimation function enables to set the threshold for switching among far-edge mobile devices, mobile edge servers and CC servers. Although it is easy to estimate the resource consumption of F_i having maximum file size set by application designer but this approach is not feasible because of varying data rates. For example, some applications generate less data in size and require minimum resources hence these applications can easily run in resource minimal far-edge devices. In addition, the resource consumption varies according to internal data structures and computational methods of application components. The dynamic threshold estimation scheme, implemented using $estimateResources()$, helps establish the switching points for dynamic execution of

MECC applications (see Figure 4.5). The data stream offloading algorithm calculates the resource consumption ($B_{used}, C_{used}, M_{used}$) after every successful execution in far-edge device. It calls update function to update the list of executions for threshold estimation.

The scheme computes two thresholds for average T_{avg} and maximum T_{max} resource consumption (see eq. 4.3, eq. 4.4, eq. 4.5, and 4.6). The T_{avg} and T_{max} are computed for last 100 successful executions of data mining components in far-edge device. The T_{avg} is computed on the basis of average file size, memory consumption, CPU consumption, and battery power consumption. Similarly the T_{max} is computed to show the maximum of file size and its corresponding resource utilization in previous 100 executions.

$$List \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_{100} \end{pmatrix} = \begin{pmatrix} SF_1 & B_1 & C_1 & M_1 \\ SF_2 & B_2 & C_2 & M_2 \\ \vdots & \vdots & \vdots & \vdots \\ SF_{100} & B_{100} & C_{100} & M_{100} \end{pmatrix} \tag{4.3}$$

$$T_{avg}[] = \{\frac{\sum_{t=1}^{100} SF_t}{100}, \frac{\sum_{t=1}^{100} B_t}{100}, \frac{\sum_{t=1}^{100} C_t}{100}, \frac{\sum_{t=1}^{100} M_t}{100}\} \tag{4.4}$$

$$mSF = max_{t=1}^{100}(SF_t) \tag{4.5}$$

$$T_{max}[] = \{mSF, B(mSF), C(mSF), M(mSF)\} \tag{4.6}$$

Assumptions are made because an MDSM application working on certain data stream may easily change its execution behavior, but for uncertain data streams, T_{max} sets the switching points. Data stream offloading is enabled when the file size lies between the T_{avg} and T_{max}. However, F_i is processed for one time and in case of memory or CPU

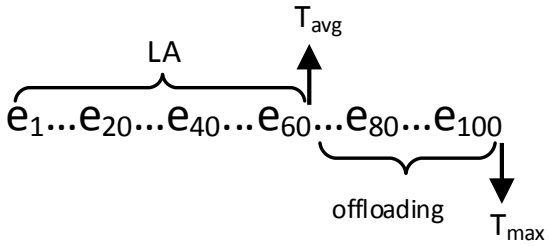


Figure 4.5: Resource Estimation Method.

exceptions, the F_i is offloaded in mobile edge server or CC servers. Otherwise, for file size less than or equal to the average file size the T_{avg} is considered as switching point. This approach helps in maximum resource utilization in far-edge device and enables data stream offloading only when the algorithm encounters a critical state like resource limitations to process the current F_i.

4.3 Adaptive Execution Model

The main objective of adaptive execution model is to facilitate low data-intensity and high compute-intensive MDSM applications. Unlike dynamic execution model that bounds the size of incoming data stream to a user-specified threshold, the adaptive execution model facilitates data stream management and offloading from multiple points during application execution. Therefore, the application designer do not need to worry about file size threshold rather the application components adaptively execute in MECC system.

4.3.1 Operations

The adaptive execution model as shown in Figure 4.6 separates the application logic and platform level logic. This approach helps in achieving generality at component level in MDSM applications. In addition, the separation of application logic lowers high coupling among application components and platform level components hence this approach helps in extendability of execution model in future. The adaptive execution model runs the multi-stage MDSM applications at foreground and platform level components perform resource monitoring, context collection, and resource estimation at each application

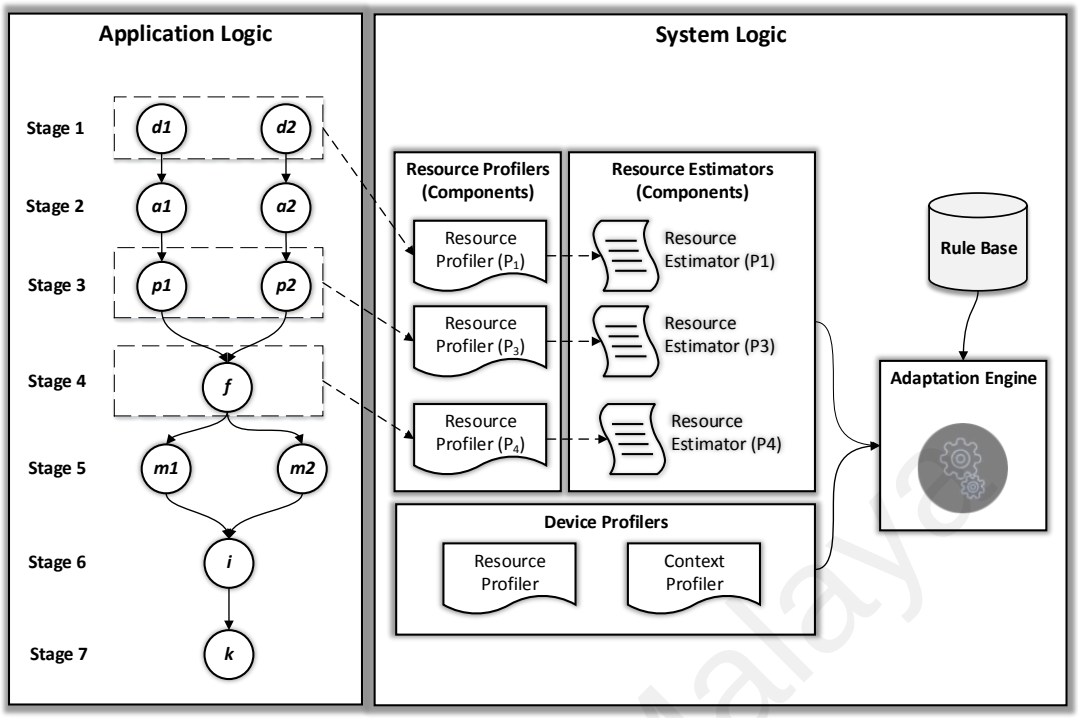


Figure 4.6: Adaptive Execution Model.

component at the back-end. Upon arrival of data stream at each component, the application switches from "Running" to "Contextualized" state in order to find the adequacy of far-edge device as execution platform. During contextualized state, execution model performs resource monitoring (to find current resource availability) and estimates the resource consumption. In case of resource availability, application components run in far-edge device otherwise execution model collects contextual information and runs rule-based scheduling strategy using *adaptation engine* component and performs data stream management and offloading operations accordingly.

Adaptive execution model performs component level adaptations whereby it performs resource profiling (in terms of battery, memory, and CPU power) of each application component. In addition, resource estimation function finds estimated resource consumption for each component. The model uses estimated resources and currently available resources for cost-benefit analysis to perform data stream offloading operations.

The component level adaptations in execution model benefits in multiple ways.

1. It enables fine-grained adaptive execution of each component.

2. The MDSM application do not need external intervention by application designers.

3. The model adapts the execution behavior easily at different far-edge mobile devices therefore MDSM applications can run using any resource constrained far-edge mobile device. In case of high resource availability, MDSM application components run in far-edge device otherwise switch to mobile edge server or CC servers.

4.3.2 Multi-point Data Stream Management

In multi-staged application execution, the nature of data stream changes at every stage. For example, in our use-case application in chapter 5, the data stream at stage 1 is available in series of raw data points collected over the passage of time. However, data adaptation operations, at stage 2, filter raw data stream and discard irrelevant information. At stage 3, data preprocessing operations convert raw data streams into feature vectors. At stage 4, data fusion operations join multiple data streams according to application requirements. At stage 5, data mining operations returns the knowledge patterns which are further integrated and summarized at stage 6. Finally, the knowledge patterns are stored at stage 7. Since the nature of operations at each stage is different therefore the computational complexity and resource requirements vary in terms of memory, CPU, and battery power consumption. On the other hand, resource dynamics in mobile devices continuously change the ground truth information about available computational and battery power resources in mobile devices. The need arises to manage the data stream at each stage when the application is in "Contextualized" state. The adaptive execution model enables five points for data management (see Table 4.1).

Table 4.1: Multi-point Data management

Point	Type of data	Reflection in use-case
P_1	Raw data	Readings from accelerometer and GPS
P_2	Filtered raw	Sliding windows of 100 readings per second
P_3	Preprocessed data	Feature extraction from each sliding window
P_4	Fused data	Event generation through feature vectors and last known GPS reading
P_5	Knowledge patterns	Activities and places

Since the data management operations are performed at multiple points therefore multiple data tables are used to store and manage the execution status of stored data files (see Fig. 4.7). Data tables are managed by global_dm and local_dm components. The global_dm components track the number of files in each data table, the local_dm generates file_identifiers and manage local data tables. The data tables are managed as follows.

Step 1: The contextualized state is invoked and the application starts data management process.

Step 2: global_dm captures the execution stage and the data files.

Step 3: global_dm invokes the relevant local_dm, updates global data table having two attributes (stage, number of files); global_dm uses stage_num to access and update the data table entries

Step 4: local_dm generates file identifiers by appending stage number with file counter such as files generated at stage 1 have the identifiers like S1-0001. Then local_dm updates

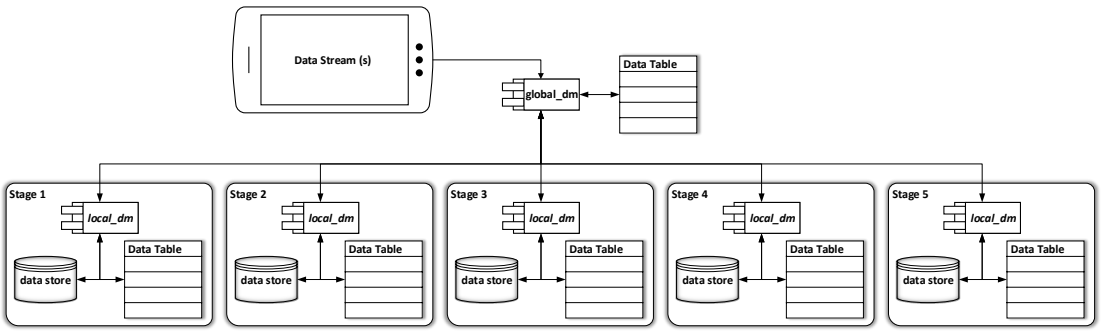


Figure 4.7: Multi-point Data Stream Management.

local data tables with two attributes (file_id, status); local_dm uses hash function to access file_id and update the status of each data file. File status is maintained as -1 for processed, 1 for under processed, and 0 for unprocessed data files.

Step 5: local_dm updates local data tables when a file is successfully processed at the next stage.

Step 6: global_dm periodically performs garbage collection by deleting processed data files from local data tables.

Step 7: local_dm updates local data tables.

Step 8: global_dm updates global data tables.

4.3.3 Multi-point Data Stream Offloading

The *adaptation engine* performs cost-benefit analysis over device capability, current resources, and contextual information. The *adaptation engine* executes multi-point data stream offloading schemes in following steps.

- Step 1: The MDSM application collects data streams from one or more data sources.
- Step 2: Resource profiler monitors the available computational and battery power resources at multiple points such as P_1, P_3, and P_4 as shown in Figure 4.6.
- Step 3: Resource estimator profiles and estimates the resource consumption at the same points.
- Step 4: *adaptation engine* performs the cost benefit analysis at each point and sets the execution mode accordingly.
- Step 5: In case of local analytics, the application initiates onboard components and performs the required operations in far-edge mobile device.

- Step 6: In case of collaborative or cloud-based analytics, the *adaptation engine* initiates multi-point data stream management process.

- Step 7: *adaptation engine* scans all communication interfaces and searches for connected mobile edge servers in the local Wi-Fi routers.

- Step 8: If local devices are found, the *adaptation engine* inquires about available computational and battery resources from connected mobile edge servers and performs cost benefit analysis for offloading.

- Step 9: If offloading is feasible in connected mobile edge servers, the *adaptation engine* offloads the data stream and initiates required application components in mobile edge servers. For example, it initiates P_2 if it offloads data stream from P_1, and initiates P_4 if data stream is offloaded from P_3.

- Step 10: If mobile edge server successfully executes the data stream with subsequent operations and returns the required results, the *adaptation engine* deletes the processed data files and updates corresponding data tables accordingly.

- Step 11: If mobile edge server does not returns results within specific time period (see eq. 4.7), the *adaptation engine* repeats step 6 to step 10 until there is no mobile edge server or required resources are not available at connected mobile edge servers.

$$t = 1.5 * avg_{makespan} \tag{4.7}$$

whereby $avg_{makespan}$ represents the average of makespan taken by last 100 successful executions.

- Step 12: In case the *adaptation engine* could not perform the offloading in mobile edge servers, it offloads the data stream in cloud environments and initiates the

required cloud services which perform the remaining operations from the point of offloading.

- Step 13: *adaptation engine* synchronizes the results with cloud services periodically and performs garbage collection and data table management accordingly.

4.3.3.1 Adaptive Resource Estimation

The *resource estimator* component collects the resource information of 100 recent executions of corresponding components inside far-edge mobile device and estimates a threshold $T_{avg}[]$ that calculates the average resource consumption in terms of memory, CPU, and battery power. Considering the threshold $T_{avg}[]$ at each point, the algorithm calculates resource consumption $R_{KB}[]$ per KilloByte (KB) using eq. 4.8. The algorithm further estimates the required resources $R_{req}[]$. The estimated threshold values are used to define the rules for cost-benefit analysis.

$$R_{KB}[] = \{\frac{T_{avg}[].B}{T_{avg}[].FileSize}, \frac{T_{avg}[].C}{T_{avg}[].FileSize}, \frac{T_{avg}[].M}{T_{avg}[].FileSize}\} \tag{4.8}$$

$$R_{req}[] = R_{KB}[].B \times currentFileSize, R_{KB}[].C \times R_{KB}[].M \times currentFileSize \tag{4.9}$$

4.3.3.2 Rule-based Scheduling

The cost benefit analysis at step 4 is performed to articulate the favorable points of data stream offloading. The adaptive execution model uses a rule-based scheduling strategy whereby the rules for transitions among far-edge devices, mobile edge servers, and CC servers are defined using a *rule base* component. Some selected rules are presented in Table 4.2. For experimentation in this thesis, 37 rules (as presented in Appendix B) were defined in the *rule base* and different rule combinations were used to formalize the logical

Table 4.2: Selected Rules for Scheduling (Please refer Appendix B for a complete list of rules).

Rule	Action
$F_{battery} \leq 10\%$	Perform data collection.
F != charging	Device not charging, check battery capacity.
F != locked	Device not locked, check application status.
UA_i == 'running'	Application is running, perform data processing.

operations in the scheduling strategy. The logical statements are based on different rules combinations using logical conjunction ($\wedge$), disjunction ($\vee$), and not ($\neg$) operators.

4.4 Summary

In chapter 4, we presented the preliminary discussion on the MDSM application architecture and its state transition model. In addition, it presented the types of data-intensive and compute-intensive applications in MECC systems. In order to address the issue of state explosion problem and seamless application execution in MECC systems, the chapter presented two execution models. The dynamic application execution model facilitate data-intensive applications having low computational complexity and minimum computational requirements. However, the application designer need to intervene in order to set the file size threshold for dynamically switching among far-edge mobile devices, mobile edge servers, and CC servers. In addition, the chapter presented adaptive execution model for MDSM applications whereby resource profiling, monitoring, and estimation is performed at multiple application components. This approach helps in adaptively switching among far-edge mobile devices, mobile edge servers, and CC servers from multiple points. Chapter 5 presents the performance evaluation of proposed execution models, discusses the experimental results, and compare with conventional static execution models.

CHAPTER 5: PERFORMANCE EVALUATION

"...the best way to show that a stick is crooked is not to argue about it or to spend time denouncing it, but to lay a straight stick alongside it..."

D.L. Moody

Chapter 4 presented the dynamic and adaptive execution models for MDSM applications. The dynamic execution model facilitates low complexity data-intensive applications. However, adaptive execution model facilitates execution of high complexity MDSM applications with low data-intensity. This chapter presents the performance evaluation of proposed execution models and compares the results with static execution models in terms of memory utilization, battery power consumption, makespan, data reduction, and accuracy of classifiers.

The chapter is outlined as follows. Section 5.1 highlights the motivational scenario for use of personal data mining in real-world and presents a use-case application for activity detection in MECC environments. Section 5.2 provides details about development and experimental setup as well as evaluation metrics. Section 5.3 presents performance results of static, dynamic, and adaptive execution models. Sections 5.3.1, 5.3.2, and 5.3.3 shows the performance results for benchmarking the static execution models for comparison. Performance results of dynamic and adaptive execution models are presented in section 5.3.4 and 5.3.5. Section 5.4 presents the comparative discussion of proposed models with static models and section 5.5 summarizes this chapter.

5.1 Use-case Application for Mobile Activity Detection

In a smart city scenario, participatory sensing applications aid in collecting data streams from citizens and sensing systems deployed on roads, railway tracks, shopping and parking areas, and countless other places in the cities. Let us consider an example of citizen

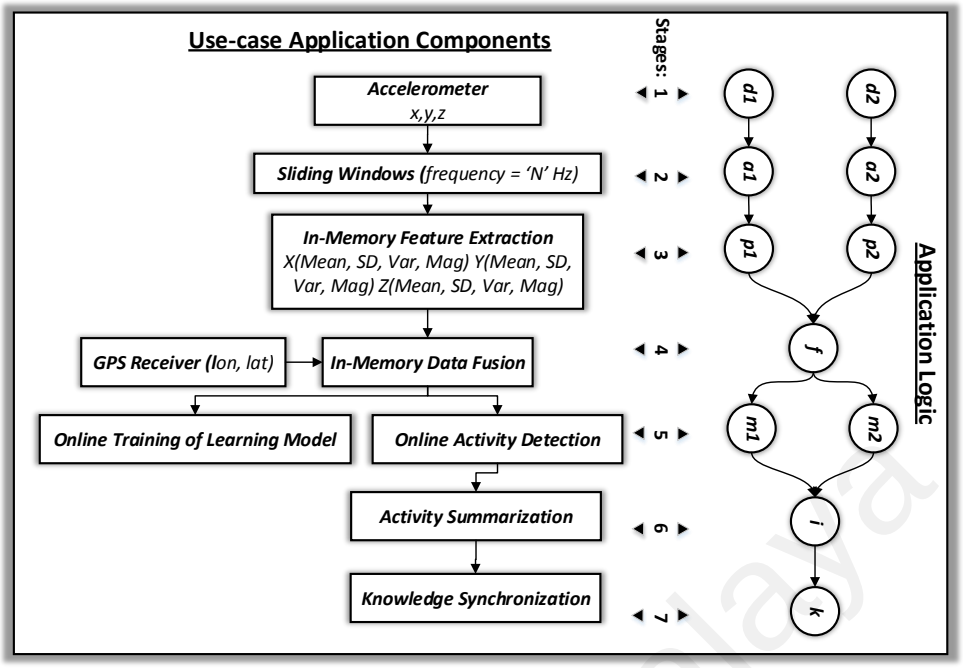


Figure 5.1: Mapping Online Activity Detection Application with Multistage MDSM Application as Presented in Section 4.1.2

sensing application for smart city whereby the city administration want to improve the quality of leisure time that citizens want to spend in public parks, sporting places, and shopping malls. The city government asks the citizens to share information about their physical activities and locations in order to improve the parking spaces and other services in crowded areas. Conventionally, the applications installed on citizens' mobile phones collect the sensing information (e.g. readings from accelerometers, GPS, nearer Wi-Fi) and transfer the whole data streams in CC systems. Using the proposed architecture, citizens first perform activity detection using MDSM components in MECC system and append the GPS coordinates for location information. The resultant knowledge is then transferred to city government's system for further utilization.

Figure 5.1 maps the use-case application components to the MDSM application architecture. The data acquisition component enables to collect the data stream from accelerometer and GPS sensors. The accelerometer produces three values for x-, y-, and z-axis for each reading. The data adaptation component enables overlapping sliding win-

dows whereby the data stream collection frequency is set as 'N' number of readings per second. The 'N' value determines the generic setting of windows size that helps in configuring different experimental settings during experiments. However, the sliding windows contain 50% overlapping values where 25% values from the start point overlaps with previous window and 25% values at the end overlaps with next window. The data preprocessing components enable feature extraction methods whereby four statistical features (*i.e.* mean, standard deviation, variance, and magnitude) are computed against each axis. The data preprocessing component generates 12 attributes for each sliding window. The data fusion component appends the latest GPS reading (*i.e.* longitude and latitude) with computed features and generates new feature vector. The data mining component enables to label the feature vectors both manually (by user intervention) and automatically (through online data annotation application). The labeled data streams represents the event data streams whereby learning models are generated and data mining component performs the activity detection for newly unlabeled data streams. The knowledge integration component produces the summary of activities and finally the knowledge management component synchronizes the produced knowledge with smart city participatory sensing applications.

5.2 Development and Experimental Setups

Since the application logic is distributed among mobile devices, mobile edge servers, and CC servers, therefore, multiple application development platforms were selected in order to evaluate the performance of execution models.

5.2.1 Development

For MDSM application execution in far-edge mobile devices, the data acquisition and adaptation, knowledge discovery, knowledge management, and system management modules of UniMiner were developed using Android SDK and Java 8 (Developers, 2014).

For collaborative application execution using mobile edge servers, the AllJoyn framework (Framework, 2015) was integrated for device discovery, P2P network formation, and data offloading. However the mobile data stream mining modules were implemented using Android SDK and Java 8. For cloud based application execution, the multi-threaded cloud services, for data stream synchronizing, data stream mining, knowledge management, and garbage collection, were developed and deployed in cloud environment using Google's compute engine (Ciurana, 2009).

5.2.2 Experimental Setup

In order to evaluate the performance of execution models for heterogeneous MDSM applications, the experimental setup varies in multiple ways. To cater the heterogeneity using data acquisition components, two data sources (*i.e.* accelerometer and GPS sensor) were used. To handle the heterogeneity using data adaptation components, sliding windows size was set as generic so that the execution models could be tested with different stream size. The heterogeneity at data preprocessing level is handled using multiple features extraction methods (such as mean, standard deviation, variance, and magnitude). The heterogeneity at data mining level is handled using three different classification algorithms namely, J48, naive Bayes, and random forest (Agrawal & Srikant, 1994; Lowd & Domingos, 2005; Breiman, 2001). The algorithms were selected due to popularity and wide acceptance in data mining research. The selected algorithms have different computational complexities hence performance variation in execution models could be measured. Similarly, the heterogeneity for knowledge management is handled by storing knowledge patterns in mobile devices and CC servers.

The experiments were performed in two phases. In the first phase, the data collection and activities annotation is performed in order to develop the learning models for activity detection. In the second phase, the experimental setup were reconfigured multiple times

in order to assess the performance of execution models. In the first phase, 12 graduate students from University of Malaya were recruited for 15 days. The users were explicitly briefed about the data collection process and instructed to keep mobile devices in right pocket of pants. For data stream collection and annotation, a separate application was developed for android phone users whereby the users can annotate 12 ambulatory activities. The application executes data acquisition, data adaptation, data preprocessing, data fusion, and data mining components of MDSM application. The activities include walking, running, standing, going upstairs, coming downstairs, sitting, going up using escalator, coming down using escalator, lying, and traveling through vehicle. The details about devices and volume of data stream are presented in Table 5.1. The collected data streams were used for learning model generation using three mobile devices (see Table 5.2).

Table 5.1: Devices and Data Collection Details.

User	**Gender**	**Model**	**Operating System**	**Collected Files**	**Size of data**
1	Female	Lenovo A6000	Android 4.4	1983	71.83 MB
2	Male	Samsung GT-18552	Android 4.1.2	3753	123.28 MB
3	Male	Samsung S3	Android 4.1.2	2679	96.99 MB
4	Male	Sony Xperia Z3	Android 5.1.1	2562	85.93 MB
5	Male	Lenovo S850	Android 4.4.2	1950	74.69 MB
6	Male	LG G3	Android 4.4.2	1237	38.94 MB
7	Male	Xiaomi Redmi2	Android 4.4	1873	63.01 MB
8	Male	Xiaomi Redmi2	Android 4.4	1135	46.40 MB
9	Male	Xiaomi Mi5	Android 5.1.1	1008	35.99 MB
10	Male	Asus Zenphone4	Android 4.3	853	39.61 MB
11	Male	Samsung Galaxy Note II	Android 4.1	569	22.75 MB
12	Male	Samsung Galaxy Tab S2	Android 5.1.1	1331	45.20 MB
Total Data				20933	744.62 MB

Table 5.2: Selected Devices for Performance Evaluation at Phase 2.

	D1	**D2**	**D3**
Make	Samsung	Sony	Samsung
Model	GT-18552	Xperia Z3	Galaxy Tab S2
Operating System	Android 4.1.2	Android 5.1.1	Android 5.1.1
Processor	1.2 GHz Quad Core	2.5 GHz Quad Core	1.9 GHz Quad Core + 1.3 GHz Quad Core
Memory	1 GB	3 GB	3 GB
Battery Power	2000 mAh	3100 mAh	4000 mAh

5.2.3 Evaluation Metrics

The performance evaluation is made in terms of: 1) battery power consumption, 2) memory utilization, 3) makespan, 4) data/bandwidth reduction, and 5) accuracy of MDSM applications. Mobile devices operate in resource-constrained environments therefore battery power consumption, memory utilization, and makespan are key performance evaluation metrics. For performance measurement, an open source software-based power profiling tool, namely Power Tutor (Gordon et al., 2013), was integrated into execution models. The profiler measures the battery power consumption in milli Watts (mW) units and memory consumption in terms of Mega Bytes (MBs). In addition, it measures the makespan in terms of milliseconds (ms). However, the process of measuring makespan differs in each execution model. For example, during static execution in Far-edge mobile devices, the makespan corresponds to average makespan that an MSDM application consumes to execute application components for each sliding window. The makespan in F2F and F2C communication models also include execution time in far-edge device as well as the round trip time between mobile devices, mobile edge servers, and CC servers. In order to achieve the data reduction objective for minimum bandwidth utilization, the amount of data stream processed in mobile devices and mobile edge servers is measured as the ratio of reduced data and overall data stream. Finally, the accuracy of classifiers is presented for each execution model.

5.3 Experiments

This section presents the results and analysis of experiments conducted in second phase. Three different types of MDSM applications were developed for experimental evaluation whereby App1, App2, and App3 use J48, naive Bayes, and random forest, respectively for activity classification. The static execution models, as of used by other similar systems such as open mobile miner (OMM), pocket data mining (PDM), and context-aware real-

time data analytic platform (CARDAP), are evaluated using three communication models. First communication model enables only far-edge mobile devices as application execution platform. The second model enables F2F communication model and the third model facilitates application execution using F2C communication model.

5.3.1 Static Application Execution using Far-edge Devices

Although Chapter 3 presented the performance of MDSM applications using Far-edge mobile devices. However, further evaluation and analysis are needed in order to compare the results of static execution models with dynamic and adaptive models.

Experimental Procedure: The experiments were performed using standalone far-edge mobile devices whereby all communication interfaces were disabled and all application components were executed using onboard resources. The learning models for each device were trained on the same device using data set collected in first phase of experiments. The experiments were conducted by two users with three mobile devices for 10 days. The mobile phones were put in right pocket of pants during application execution.

Findings: As witnessed during problem analysis in Chapter 3, the mobile devices failed to execute MDSM applications for data streams with large size. Therefore, in order to lower the impact of data size, the initial data rate was settled as 100 readings per second. Later on the time interval is increased by 10 milliseconds. Despite increasing the time interval, MDSM applications on D1 failed to execute a few times due to low memory availability and high data rates. However, the MDSM applications on devices D2 and D3 never failed and completely executed the data streams with high data rates as well. It was observed that the MDSM applications became stable when time interval reached 70 ms.

Figure 5.2 shows the average battery consumption trends of three MDSM applications using D1. Since the applications use different data stream mining algorithms therefore the computational complexity of each algorithm varies which in turn affects the

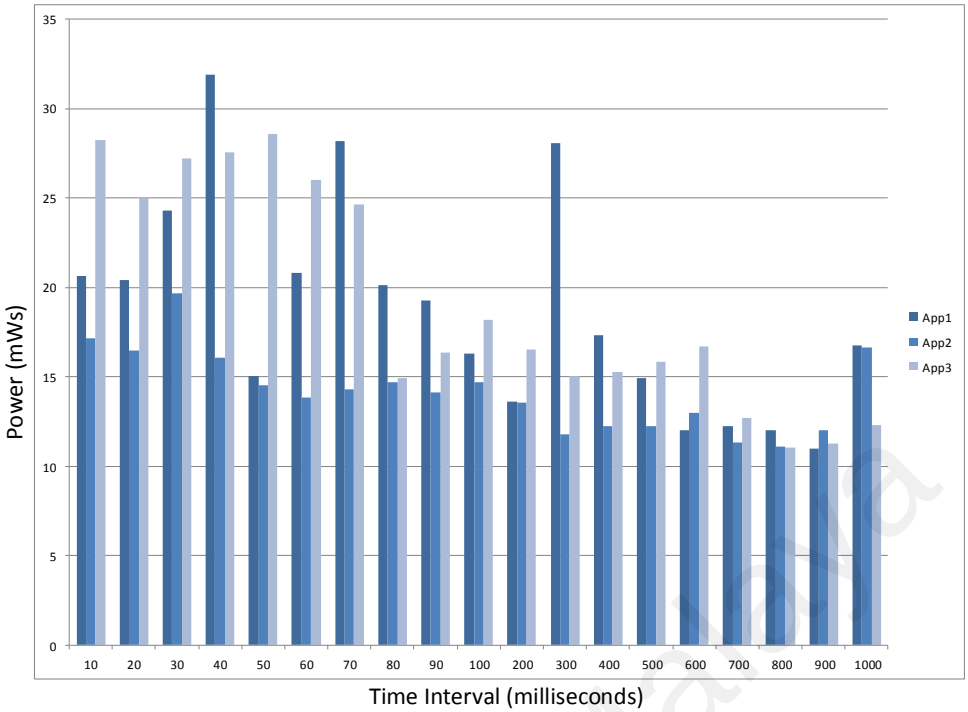


Figure 5.2: Power Consumption of D1 using App1, App2, App3.

energy consumption by MDSM applications. The variations in computational complexities emerge due to different data structures and the traversal behaviors of data stream mining algorithms. For example, App1 uses J48 algorithm which contains a tree based data structure and enables non-linear traversal behaviors. Alternatively, App2 uses naive Bayes algorithm having array data structures and linear traversal methods therefore the computational complexities of both algorithms differ. The overall trend in Figure 5.2 shows that App2 consumed comparatively lower battery power followed by App1 and App3. It was observed that overall average battery power consumed by D1 occasionally surpassed 30 mW and the average battery power consumed by MDSM applications remained about 27 mW.

Figure 5.3 shows battery power consumption trends of D2 and D3 (the results of D1 presented earlier because of different data rates). The results show the minimum, average, and maximum battery power consumed by MDSM application components. Since the applications' performance varies due to computational complexity and traversal be-

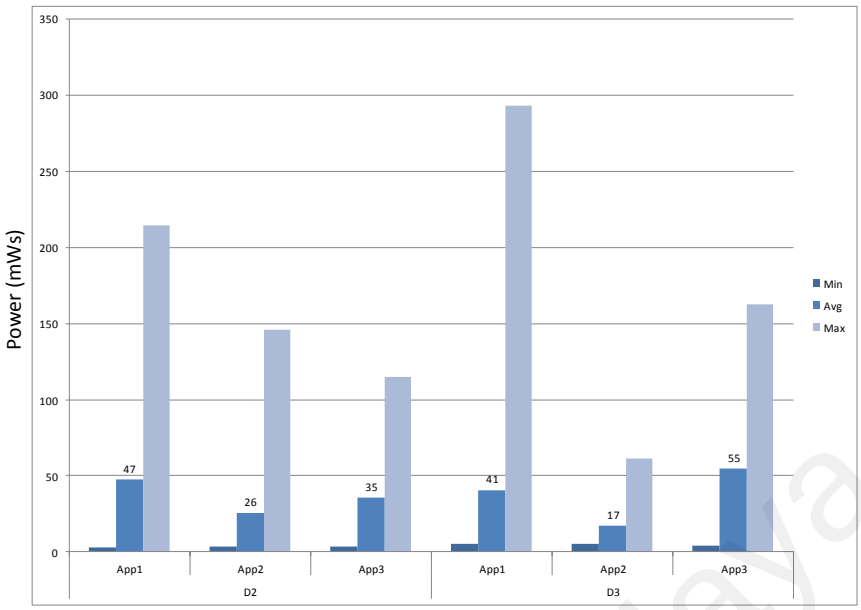


Figure 5.3: Power Consumption of D2 and D3 using App1, App2, App3.

haviors of data stream mining algorithms. Therefore, using D2, App1 consumed 47 mW on average as compared to App2 and App3 that consumed 26 mW and 35 mW, respectively. The same battery power consumption trends were witnessed using D3. Overall, D2 performed comparatively better because of availability of multi-core processors that enable task parallelization and efficient application execution in mobile devices.

Considering all other sources of battery power consumption as constant, the battery depletion time of MDSM applications is presented in Table 5.3. The time for complete battery depletion from 100% charge is derived using eq. 5.1. The results were calculated over a total battery capacity (in milliampere hours) and voltage level of 3.7V (volts). The 'P' value was determined from battery power consumed (in mW) by MDSM applications and the *total_battery_capacity* value was substituted by 2000 mAh for D1, 3100 mAh for D2, and 4000 mAh for D3.

$$Time_{depletion} = \frac{total\_battery\_capacity * 3600}{(P/3.7)} \tag{5.1}$$

The average battery depletion time of D1 and D2 remained around 29 to 30 hours. Using

Table 5.3: Battery Charge Depletion Time using Far-edge Devices.

Device	**Application**	**P (mW)**	$Time_{depletion}$
D1	App1	19	24 Hours 02 Minutes
	App2	14	32 Hours 14 Minutes
	App3	20	22 Hours 22 Minutes
D2	App1	47	15 Hours 04 Minutes
	App2	26	26 Hours 45 Minutes
	App3	35	20 Hours 06 Minutes
D3	App1	41	22 Hours 06 Minutes
	App2	17	52 Hours 28 Minutes
	App3	55	16 Hours 15 Minutes

D2, the time decreased around 20 hours. The low battery power consumption at D1 occurred because of low data processing due to limited memory availability. Alternately, D3 consumed low battery power because of availability of multiple processors which enable fast process execution. Hence, D3 does not need to store and process the data in memory buffers for longer period which results in efficient utilization of battery power. D2 consumed comparatively higher energy because of single processing unit and data management in memory buffers.

Since the computational behaviors of MDSM application differ which impact the makespan. The unprocessed data streams need to be stored in memory buffers therefore some energy spikes were observed during application execution. Figure 5.4 shows the statistical results of standard deviation (calculated using eq. 5.2) of battery power consumption trends. The results show that battery power consumption of App2 on D1 and D3 had more energy spikes as compared to App2 and App3. Alternatively, App3 on D2 had significantly high energy spikes as compared to App1 and App3.

$$standard\_deviation = \sqrt{\frac{\sum (x-\mu)^2}{(n-1)}} \tag{5.2}$$

Here 'x' represents the observed values and 'μ' is the computed average of 'n' number of observations.

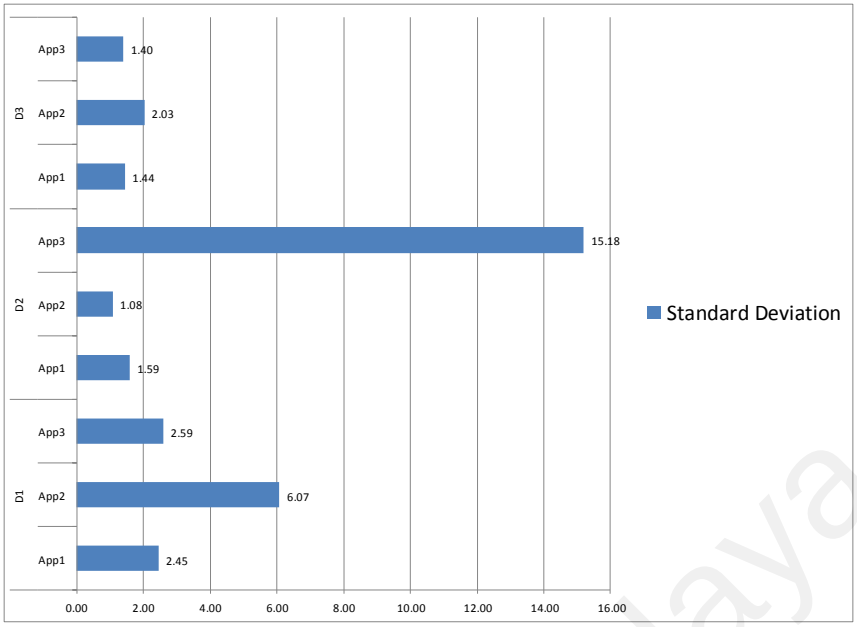


Figure 5.4: Standard Deviation in Power Consumption

The memory consumption trends of MDSM applications using D1 are presented in Figure 5.5. It was observed that the overall memory consumption of MDSM applications remained similar, *i.e.* 26.77 MB for App1, 27.30 MB for App2, and 26.77 MB for App3. However, the computational behaviors and the data rates have slight and ignorable impact on memory utilization (see Table 5.4). Figure 5.6 shows the memory utilization of MDSM

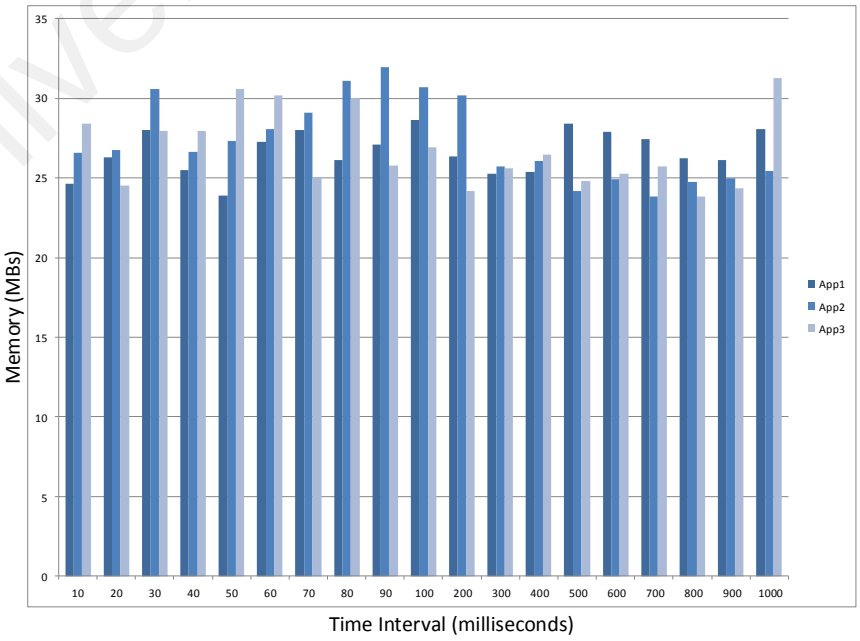


Figure 5.5: Memory Consumption of D1 using App1, App2, App3.

Table 5.4: Impact of Data Rate on Memory Utilization in D1.

Data Rate (ms)	App1 (MBs)	App2 (MBs)	App3 (MBs)
10	0	0	1.63
20	0	0	0
30	1.34	3.26	1.15
40	0	0	1.17
50	0	0.04	3.82
60	0.63	0.74	3.39
70	1.34	1.8	0
80	0	3.77	3.21
90	0.41	4.66	0
100	1.95	3.42	0.16
200	0	2.87	0
300	0	0	0
400	0	0	0
500	1.75	0	0
600	1.26	0	0
700	0.79	0	0
800	0	0	0
900	0	0	0
1000	1.39	0	0

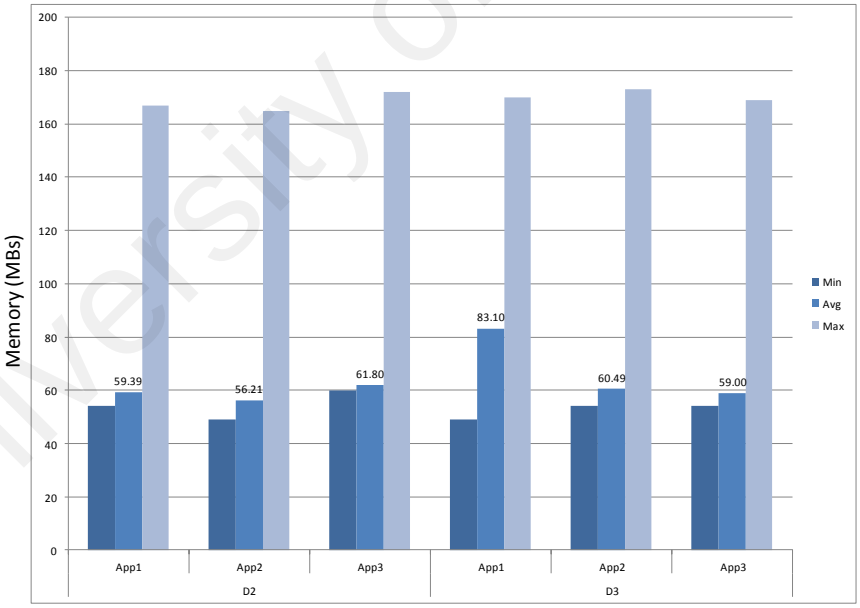


Figure 5.6: Memory Consumption of D2 and D3 using App1, App2, App3.

applications using D2 and D3. The results show that on average D2 consumed 59.13 MB and D3 consumed 67.53 MB as compared with 26.91 MB consumed by D1. The devices D2 and D3 consumed more memory because of high memory availability which enables to keep more data in memory and enable fast data processing. In application perspectives,

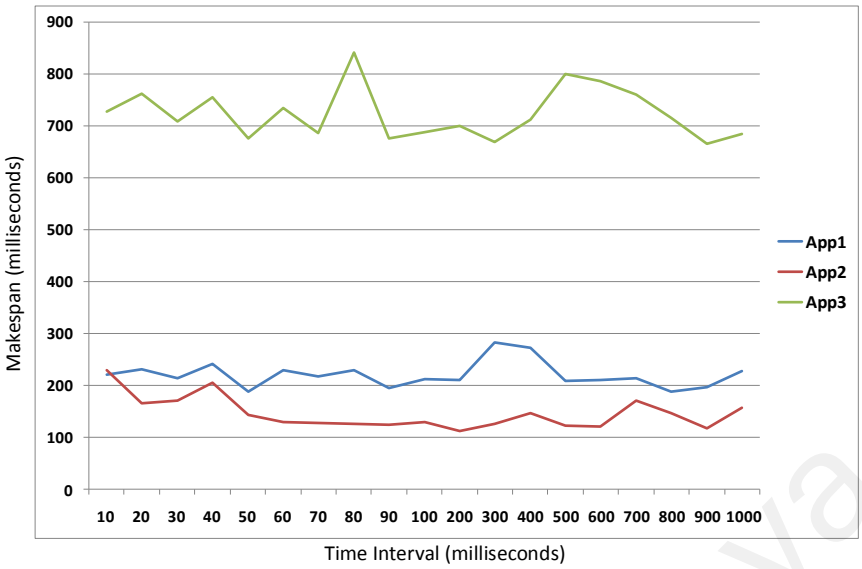


Figure 5.7: Makespan of D1 using App1, App2, App3.

App1 and App2 consumed more memory in D3 as compared to D2 but App2 consumed slightly higher memory on D2.

The makespan of MDSM applications in D1 is shown in Figure 5.7. The variation in makespan is witnessed due to different computational behaviors of MDSM application components. It was observed that App2 executes comparatively faster because of linear data structures and less computational complexities of naive Bayes classifier. However,

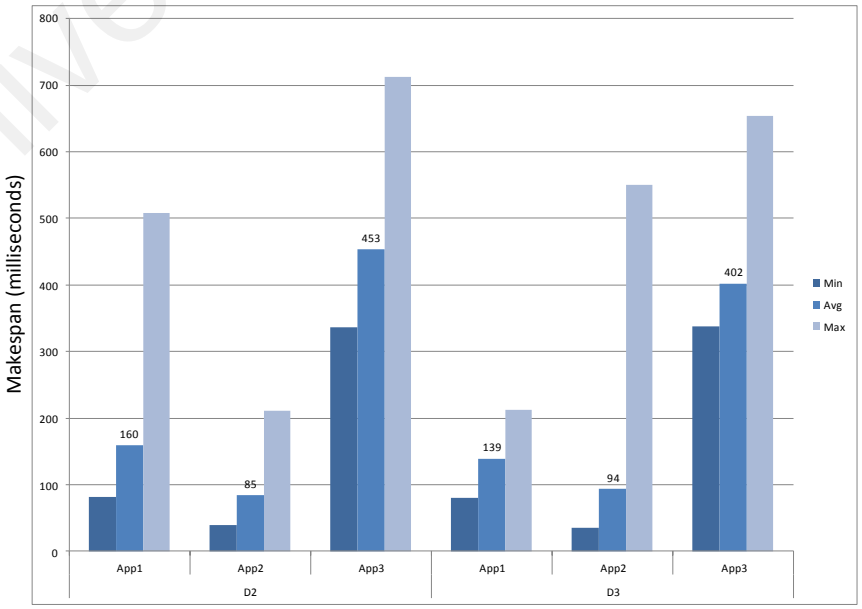


Figure 5.8: Makespan of D2 and D3 using App1, App2, App3.

non-linear data structures such as used by J48 and random forest classifiers take more time for traversals and computations. Similarly, App3 which uses multiple tree-based data structures consumes marginally higher makespan as compared to single tree-based data structure of App2. Similarly, the MDSM applications in D2 and D3 had shown the similar results (see Figure 5.8). The comparison shows that App1 on D1 consumed about 50% extra makespan as compared to App2 and in case of App3 the average makespan was increased about 600%. Similarly on D2, App1 consumed 94% extra makespan and the ratio was increased about 433% in the case of App3. Likewise, App1 consumed 48% which was increased upto 328% using App3. Overall, D2 executes MDSM applications faster as compared to D1 because of more memory availability and D3 performs better due to availability of multiprocessor architectures.

Considering the results of data stream mining algorithms, the accuracy of classifiers is calculated using confusion matrix for physical activities. Figure 5.9 shows the overall accuracy trends of MDSM applications. The results show that all applications produced more accurate results on D1, followed by D3 and D2. An interesting fact was found that

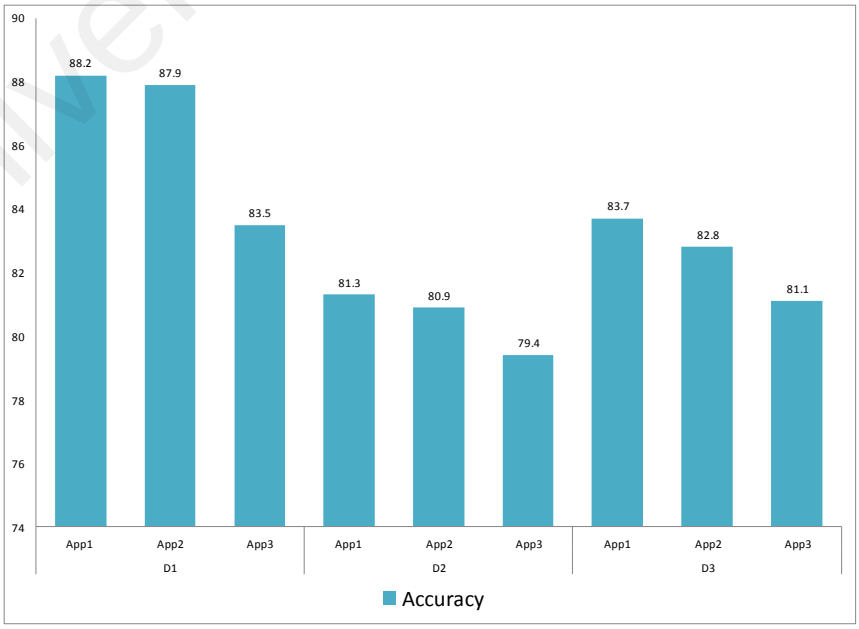


Figure 5.9: Overall Accuracy of App1, App2, App3.

single-tree based learning models (*i.e.* used by App1) provide more accurate results as compared to multiple tree-based and probabilistic learning models.

5.3.2 Static Application Execution using F2F Communication Model

Considering the three-layer architecture in Chapter 3, the MDSM applications were evaluated using F2F communication model.

Experimental Procedure: The experiments were performed using four far-edge mobile devices. In this experiment, D3 was replaced with D4 and D5 (see details in Table 5.5). The devices can function as both client and server, but not at the same time. Therefore, two versions of each MDSM application were developed to run on far-edge mobile client devices and mobile edge servers. The client side application enabled data acquisition, data adaption, device discovery, peer network formation, data offloading, knowledge integration, and knowledge management components. The client device collects the data streams, and sends to mobile edge servers which perform feature extraction, data fusion and data mining operations. The resultant knowledge patterns were sent back to client devices using pattern synchronization components and client devices perform knowledge integration and knowledge management operations. To further the experiment, the learning models were reused as previously developed using far-edge mobile devices. Two mobile users performed the experiments whereby the client device was placed in the right pocket of the pants and the mobile edge server in the left pocket. The experiments were run for three days whereby the users were asked to run each application for 24 hours. Since the communication model uses local communication channels therefore only the Wi-Fi interfaces of all devices were turned on during experiments. All other factors of complexity such as mobility, resource availability in mobile edge servers, and battery power consumption by other processes in client and server devices were neglected during these experiments.

Table 5.5: Selected Mobile Edge Servers.

	D4	**D5**
Make	Samsung	Redmi
Model	Galaxy S7 Edge	Mi4i
Operating System	Android 6.0	Android L
Processor	Quad-core 2.3 GHz Mongoose + Quad-core 1.6 GHz Cortex-A53	Qualcomm Snapdragon 615 Octa-core 64-bit
Memory	4 GB	2 GB
Battery Power	3600 mAh	3120 mAh

Findings: The measurements were taken from client mobile devices in order to evaluate the resource consumption during data communication between client and server devices. It was observed that, unlike direct execution in mobile devices, the MDSM applications did not crashed because the data streams were partially processed in client devices and partially in mobile edge servers. The experiments were performed by setting the time interval for data rate as 10 ms.

On both devices, *i.e.* D1 and D2, similar battery power consumption trends were observed (see Figure 5.10) whereas App1 consumed comparatively higher battery power.

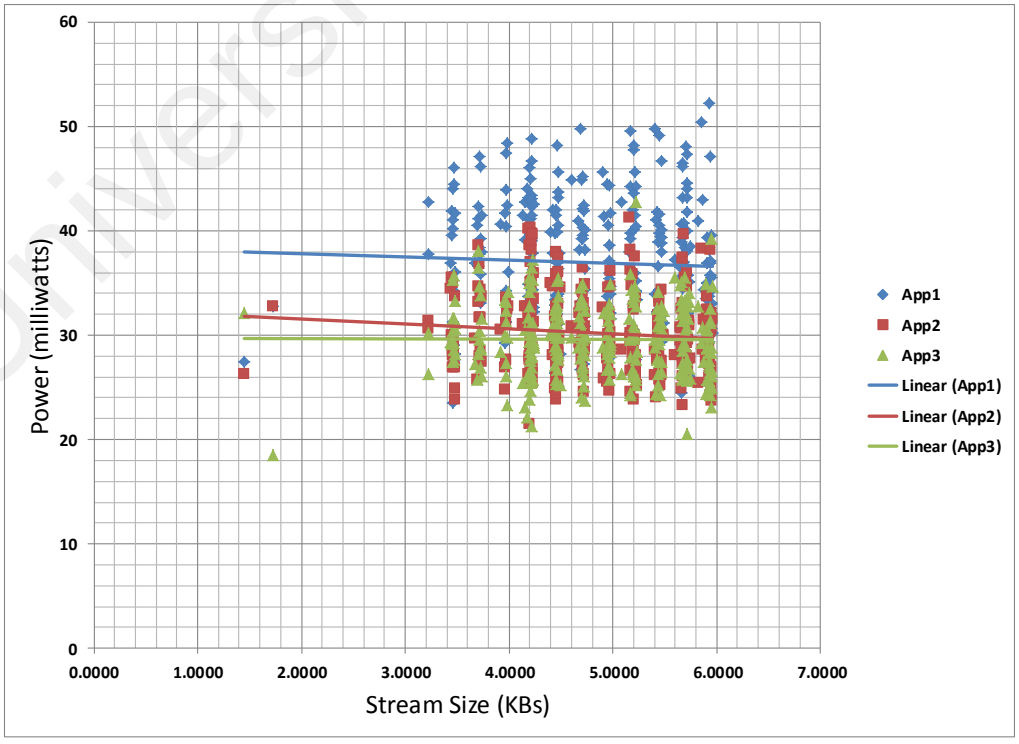


Figure 5.10: Average Power Consumption of App1, App2, App3.

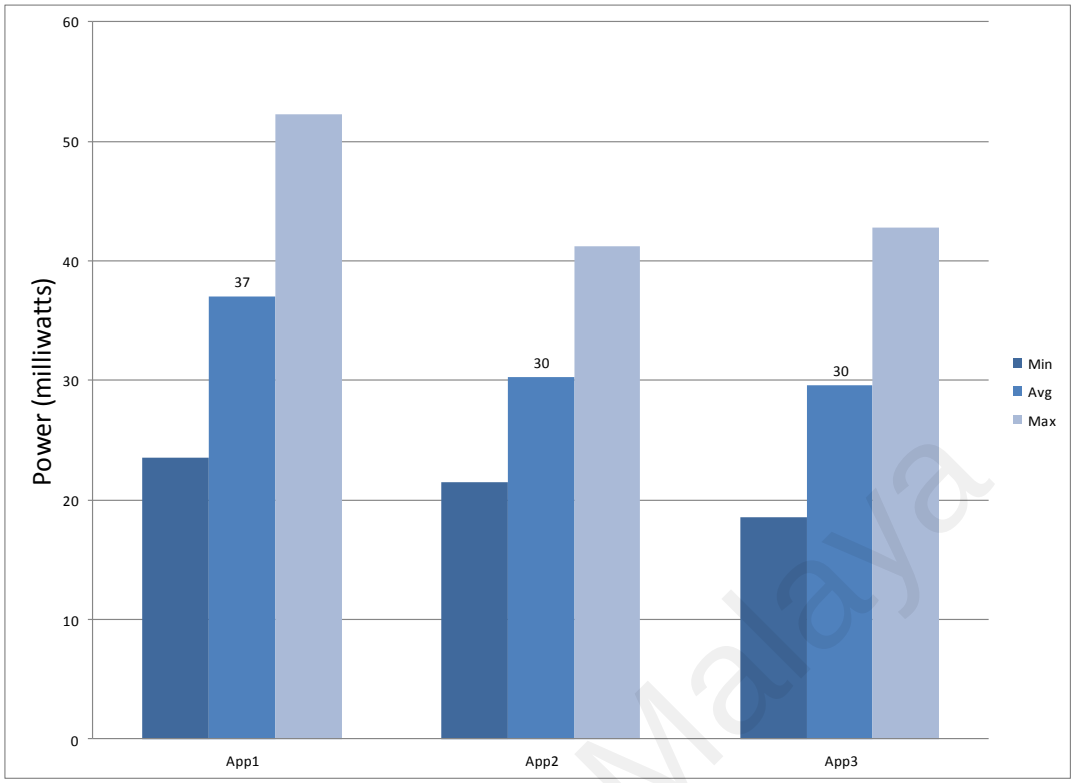


Figure 5.11: Power Consumption Trends of App1, App2, App3.

It was observed that size of data stream did not affected the battery power consumption significantly. Figure 5.11 shows the minimum, average, and maximum battery power consumed during experiments. On average, App1 consumed 37 mW for each batch of sliding windows whereas App2 and App3 consumed almost 30 mW. However, there exist variations in battery power consumption during each iteration. Considering these variations, the standard deviation was computed using eq. 5.2. The standard deviation in App1 observed as 5.91 mw, which decreased in App2 and remained 3.77 mW. However, in App3 the standard deviation remained 3.36 mW.

Considering the battery power consumption trends and standard deviation results, the battery depletion time was computed for D1 and D2 using eq. 5.1. It should be noted that total battery capacity of D1 was 2000 mAh and D2 had 3100 mAh battery. Table 5.6 shows the battery depletion time. The average battery depletion time for D1 remained between 12 to 16 hours, however, for D2, the battery can last from 18 to 24 hours. Con-

Table 5.6: Battery Charge Depletion Time using F2F Communication Model.

	D1					
	App1		App2		App3	
	P	$Time_{depletion}$	P	$Time_{depletion}$	P	$Time_{depletion}$
Min	21	21 Hrs 14 Mins	22	20 Hrs 18 Mins	15	39 Hrs 06 Mins
Avg-S.Dev	31	14 Hrs 32 Mins	25	18 Hrs 16 Mins	24	18 Hrs 50 Mins
Avg	36	12 Hrs 33 Mins	29	15 Hrs 31 Mins	28	16 Hrs 16 Mins
Avg+D.Dev	41	11 Hrs 22 Mins	33	13 Hrs 45 Mins	32	14 Hrs 37 Mins
Maximum	58	08 Hrs 05 Mins	40	11 Hrs 10 Mins	38	13 Hrs 08 Mins
	D2					
	App1		App2		App3	
	P	$Time_{depletion}$	P	$Time_{depletion}$	P	$Time_{depletion}$
Min	23	30 Hrs 32 Mins	21	33 Hrs 17 Mins	19	36 Hrs 22 Mins
Avg-S.Dev.	31	22 Hrs 20 Mins	26	26 Hrs 46 Mins	27	25 Hrs 48 Mins
Avg	37	18 Hrs 06 Mins	30	23 Hrs 34 Mins	30	23 Hrs 34 Mins
Avg+S.Dev.	43	16 Hrs 03 Mins	34	20 Hrs 24 Mins	33	21 Hrs 25 Mins
Max	52	13 Hrs 23 Mins	41	17 Hrs 19 Mins	43	16 Hrs 02 Mins

sidering the standard deviation, D1 depletes between 11 to 19 hours and D2 can last between 16 to 27 hours. On other hand, during slow battery depletion D1 lasts between 20 to 39 hours and D2 depletes between 30 to 36 hours. During fast battery depletion, D1 lasts between 08 and 13 hours and D2 depletes between 13 and 17 hours. Overall, D1 depletes fast because of low availability of battery capacity.

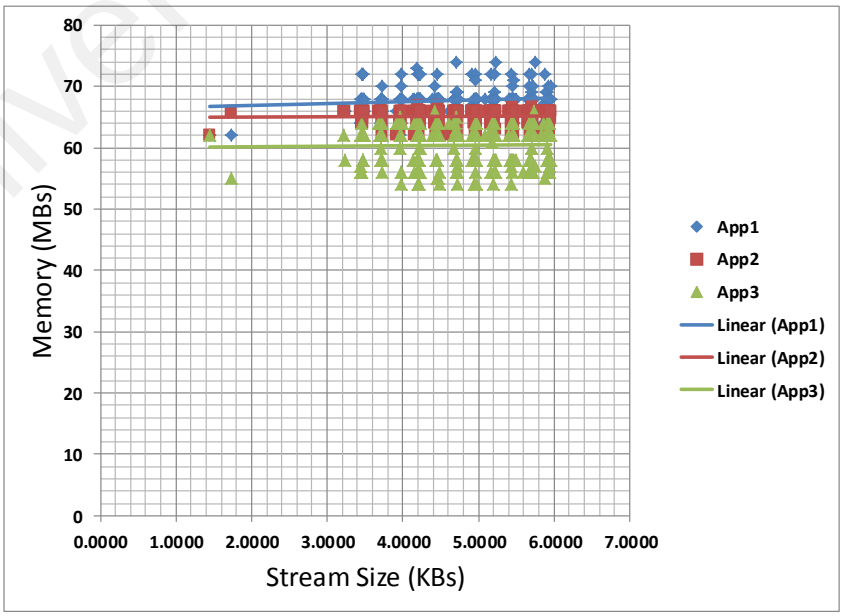


Figure 5.12: Average Memory Consumption of App1, App2, App3.

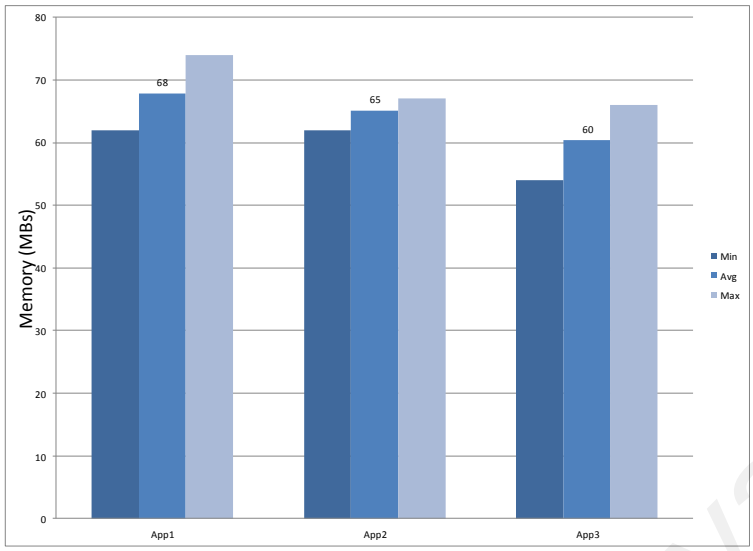


Figure 5.13: Memory Consumption Trends of App1, App2, App3.

Figure 5.12 shows the average memory consumption of MDSM applications on D1 and D2. It was observed that there is no significant variation in memory consumption during whole experiment. On average, App1 consumed comparatively higher memory, followed by App2 and then App3. Figure 5.13 depicts the minimum, average, and maximum memory consumption trends. It was observed that despite variation in memory availability, the MDSM applications consumed similar amount of memory. This behavior occurs because the computational complexity of application components in mobile devices did not varied significantly and data stream mining components were executed in mobile edge servers. The difference between lowest and highest memory consumption remained about 12 MB in the case of App1 and App3 while it lowers to 5 MB in the case of App2. However, the standard deviation in overall memory consumption remained 1.81 MB for App1, 1.34 MB for App2, and 3.02 MB for App3. Hence, the results shows the insignificant variation in memory consumption.

Since the MDSM applications use client-server based execution model, therefore, the makespan accounts for time of sending and receiving data streams and the time to execute data streams in mobile edge servers. Figure 5.14 depicts the average makespan in

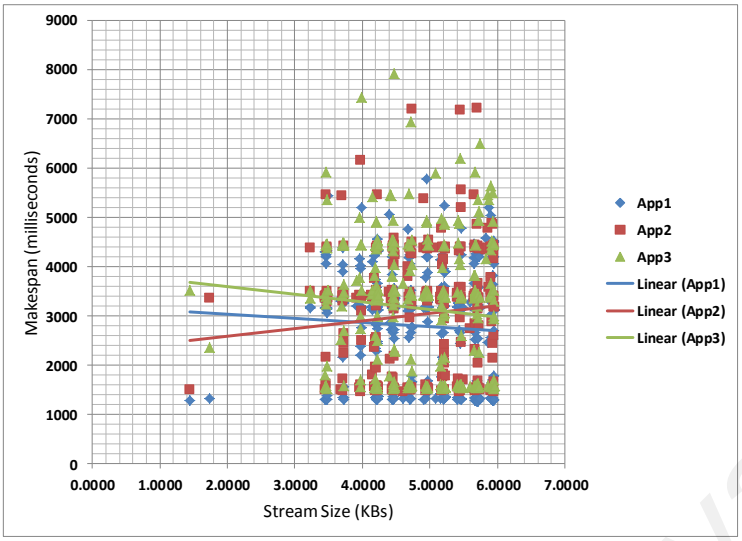


Figure 5.14: Average Makespan of App1, App2, App3.

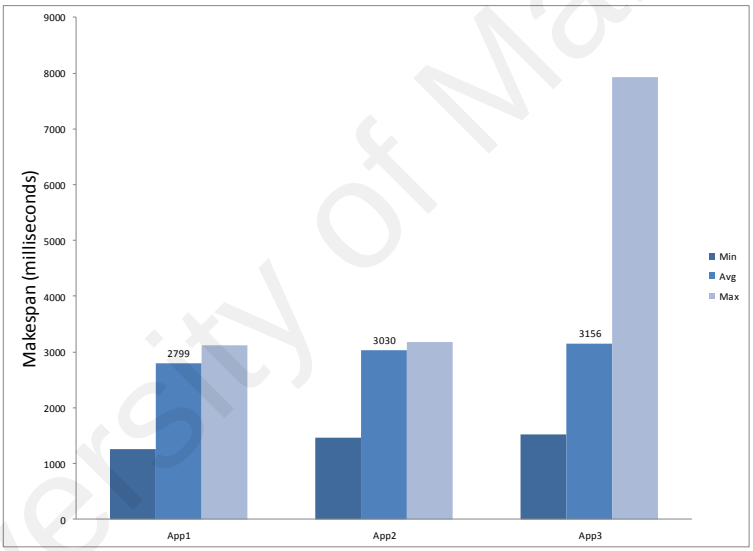


Figure 5.15: Makespan Trends of App1, App2, App3.

MDSM applications. It was observed that on average App3 had more makespan followed by App1 and then App2. However, the makespan values had great amount of variations. The computed standard deviation for makespan values of App1 remained 1121 ms which accounts about 40% of average makespan *i.e.* 2799 ms. Similarly, standard deviation of makespan for App2 was 1232 ms (about 40%) and for App3 it went upto 1347 ms (about 42%). Figure 5.15 shows the makespan trends in MDSM applications.

Considering the results of data stream mining algorithms in mobile edge servers (see

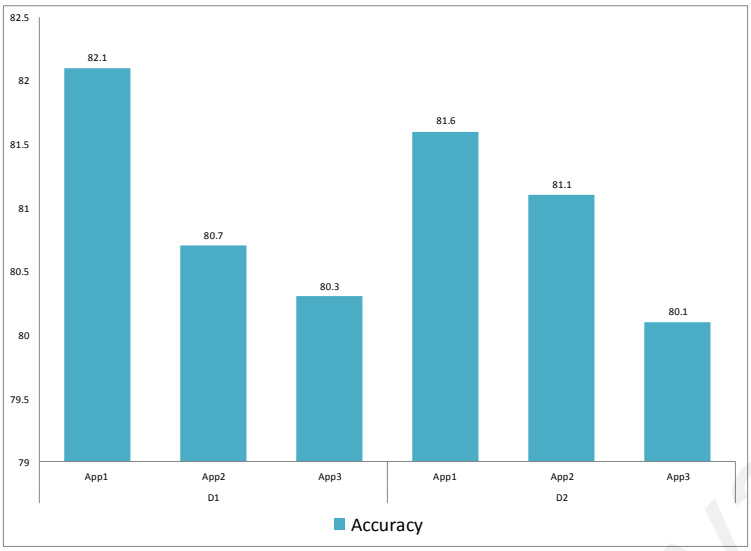


Figure 5.16: Accuracy Trends using D1 and D2.

Figure 5.16), the accuracy of the classifiers mainly depends upon the quality of learning models. During these experiments, the accuracy remained more or less similar as it was in the case of far-edge mobile devices.

5.3.3 Static Application Execution using F2C Communication Model

Referring to three-tier architecture, the MDSM applications were executed and evaluated using F2C communication model.

Experimental Procedure: The experiments were performed using D1 and D2 and Wi-Fi interfaces of both devices were enabled. To further the experiments, two versions of each MDSM application were developed. On the mobile side, an android application was developed that enables data acquisition, data adaptation, data offloading, knowledge integration, and knowledge management components. On the cloud side, the cloud services for each MDSM application were orchestrated. To this end, the cloud services for data synchronization, data preprocessing, data fusion, and data mining were deployed using Google AppEngine (Ciurana, 2009). In addition, learning models were reused and deployed on the cloud side for each MDSM application. The experiments were performed by commuting inside University network in order to seamlessly switch among different

communication points (*i.e.* wireless access points) and avoid manually connecting with Internet through other networks. The wireless access points support bandwidth utilization from 10 Mbps (Megabits per second) to 100 Mbps. The experiments were performed by two users by putting mobile devices in the right pockets of their pants. The experiment were run for three hours for each MDSM application.

Findings: In these experiments, the measurements were taken from client mobile devices in order to find the cost of data communication between mobile devices and CC servers. It was observed that, unlike MDSM application using F2F communication model, the applications occasionally crashed with high data rate. Therefore, the experiments were performed by setting the time interval of 20 ms.

The battery power consumption trends on both devices *i.e.* D1 and D2 remained similar. In both devices, App2 consumed comparatively lower battery power, followed by App1 and App3 (see Figure 5.17). It was observed that App1 and App2 remained stable however in some cases App3 consumed comparatively higher battery power.

The variations in battery power measurements were calculated by finding the standard deviation using eq. 5.2. The standard deviation in App2 was 3.25 mW which in-

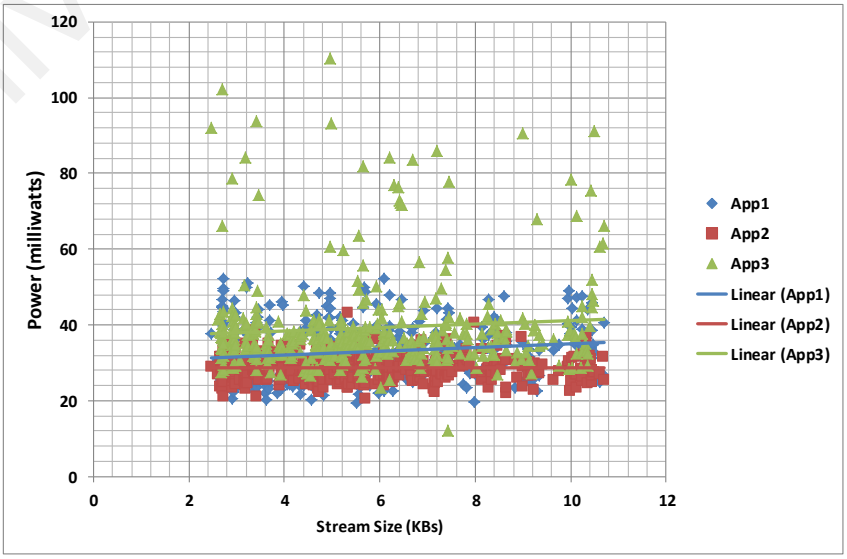


Figure 5.17: Average Power Consumption of App1, App2, App3.

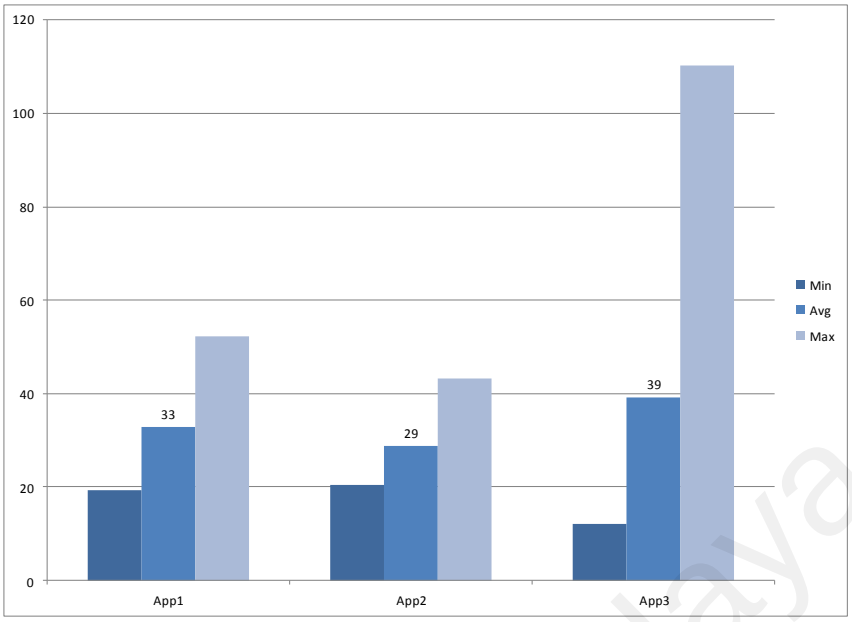


Figure 5.18: Power Consumption Trends of App1, App2, App3.

Table 5.7: Battery Charge Depletion Time using F2C Communication Model.

	D1					
	App1		App2		App3	
	P	$Time_{depletion}$	P	$Time_{depletion}$	P	$Time_{depletion}$
Min	18	25 Hrs 05 Mins	19	23 Hrs 37 Mins	11	40 Hrs 36 Mins
Avg-S.Dev	24	18 Hrs 50 Mins	25	18 Hrs 16 Mins	25	18 Hrs 16 Mins
Avg	31	14 Hrs 32 Mins	28	16 Hrs 25 Mins	38	12 Hrs 08 Mins
Avg+D.Dev	38	12 Hrs 08 Mins	31	14 Hrs 32 Mins	51	09 Hrs 10 Mins
Maximum	50	09 Hrs 28 Mins	42	10 Hrs 57 Mins	108	4 Hrs 11 Mins
	D2					
	App1		App2		App3	
	P	$Time_{depletion}$	P	$Time_{depletion}$	P	$Time_{depletion}$
Min	20	34 Hrs 41 Mins	21	33 Hrs 17 Mins	14	49 Hrs 15 Mins
Avg-S.Dev	27	25 Hrs 48 Mins	27	25 Hrs 48 Mins	28	24 Hrs 57 Mins
Avg	34	20 Hrs 24 Mins	30	23 Hrs 34 Mins	41	17 Hrs 18 Mins
Avg+D.Dev	41	17 Hrs 18 Mins	33	21 Hrs 25 Mins	54	13 Hrs 14 Mins
Maximum	53	13 Hrs 38 Mins	44	16 Hrs 04 Mins	112	06 Hrs 14 Mins

creased to 7.13 mW in App1 and 12.86 mW in App3. Figure 5.18 shows the minimum, average, and maximum battery power consumption trends. The battery depletion times of both devices were computed using eq. 5.1 and the results are presented in Table 5.7.

The battery depletion time of D1 fluctuates between 4 hours and 41 hours and for D2, the battery completely discharges between 6 to 49 hours. However the average battery depletion time of D1 remained between 12 to 16 hours and D2 discharges between 17 to

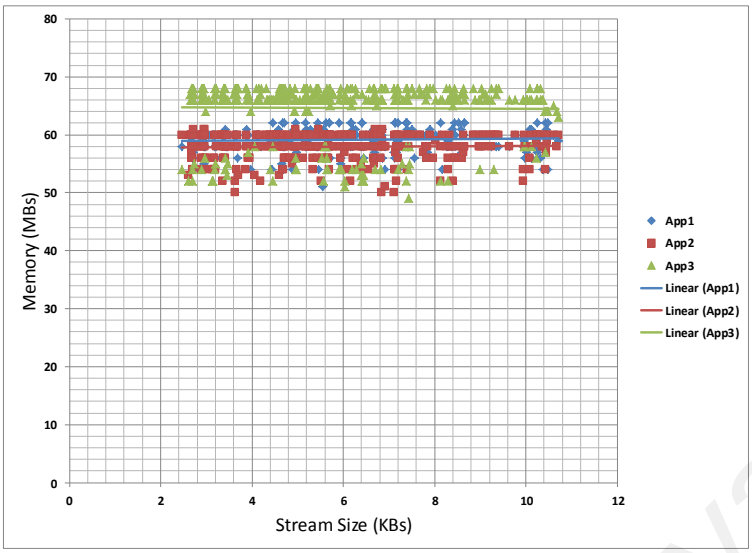


Figure 5.19: Average Memory Consumption of App1, App2, App3.

24 hours. Considering standard deviation, D1 discharges between 9 to 18 hours and D2 lasts for 13 to 26 hours.

Figure 5.19 shows the memory consumption of MDSM applications using F2C communication model. It was observed that App2 consumed comparatively lower memory followed by App1 and App3. However, there were insignificant variations during whole experiment. This behavior uncovered because computational complexities of application components in far-edge devices did not significantly varied. The minimum, average,

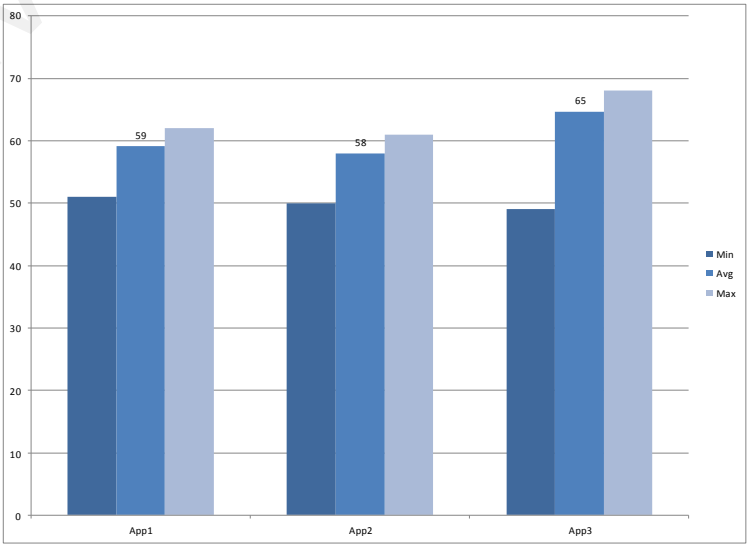


Figure 5.20: Memory Consumption Trends of App1, App2, App3.

and maximum memory consumption trends are presented in Figure 5.20. The difference between lowest and highest memory consumption remained about 11 MB for App1 and App2. However it increased upto 19 MB for App3. This behavior uncovered because MDSM applications stores data streams temporarily in memory buffers and each batch of sliding windows is sent upon the arrival of previously transferred data streams. The high makespan in app3 increased the memory consumption. The standard deviation in memory consumption remained 1.85 MB for App1, 2.24 MB for App2, and 4.64 MB for App3. This deviation is insignificant as compared to available memory in devices (*i.e.* 832 MB in D1 and 2679 MB in D2).

The makespan in MDSM applications vary because of variable bandwidths and mobility of devices. Figure 5.21 shows the average makespan of MDSM applications using F2C communication model. It was observed that on average there is insignificant difference however the variations in each set of execution is found. Figure 5.22 shows the minimum, average, and maximum makespan in MDSM applications. The average makespan in applications remained between 2.7 and 2.9 seconds. However, the difference between

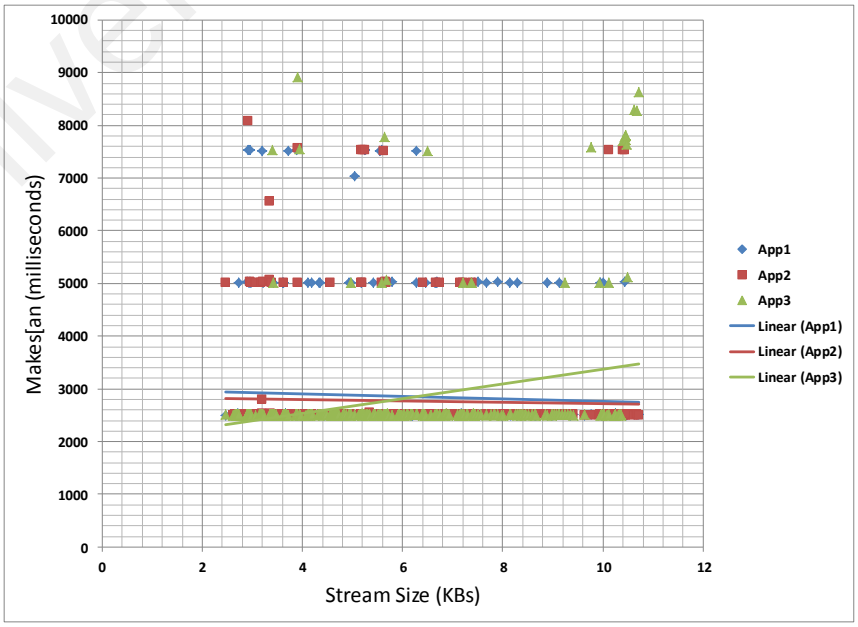


Figure 5.21: Average Makespan of App1, App2, App3.

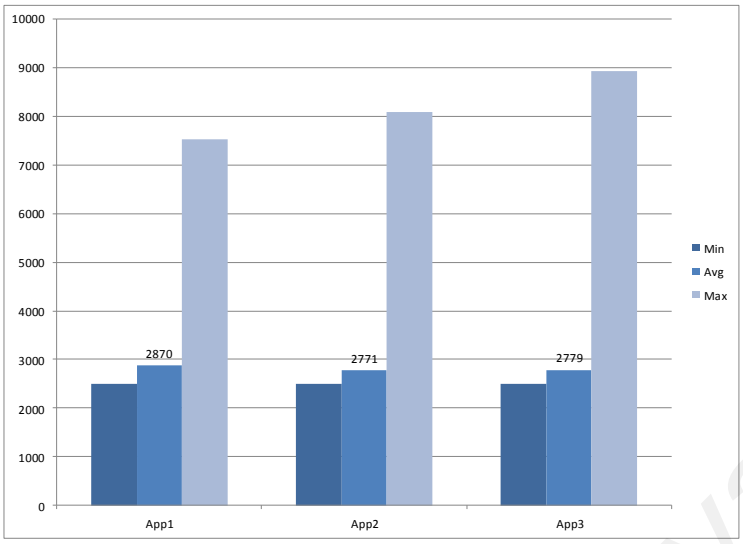


Figure 5.22: Makespan Trends of App1, App2, App3.

minimum and maximum makespan increased about 5 to 6 seconds. The standard deviation in makespan of App1 was found as 1039 ms which accounts about 36% of average makespan. For App2, the standard deviation was 947 ms (about 34%) and for App3 it was increased up to 1093 ms (about 39%). Although there was a minor variation in average and standard deviation values of makespan, however, the makespan spikes were observed due to unavailability of Internet connection or low bandwidth.

Considering the results of data stream mining algorithms, Figure 5.23 presents the accuracy values of each application on D1 and D2. Similar to previous experiments, the accuracy results were more or less shown the same trends.

The experimental evaluation proved that performance of MDSM applications vary in each device. In order to find the correlation among power consumption, makespan, and memory consumption, the statistical analysis were performed by finding the Pearson's correlation coefficients (termed as 'r') using bi-variate analysis method in SPSS (Norusis, 2008). In addition, the coefficients of determination were computed against each 'r' value in order to determine the percentage of shared variance in each correlation. The Pearson's correlation coefficients were selected instead of other correlation methods such as

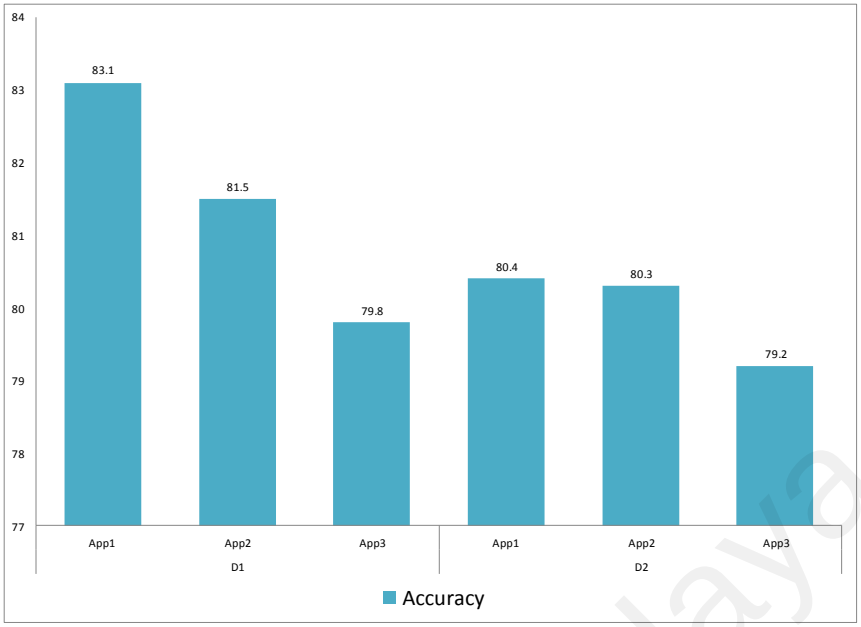


Figure 5.23: Accuracy Trends of App1, App2, App3.

Kendall's Tau-b and Spearman's correlation coefficients because it supports linear correlation among two variables (Bolboaca & Jäntschi, 2006). In addition, it works with the data which is not normally distributed and may have outliers. Moreover, Pearson correlation coefficients can be computed from ordinal (*i.e.* ordered) variables. The 'r' values were computed using eq. 5.3. It should be noted that 'r' values always lies between the interval of -1 and +1 whereas the interval and level of significance of relationship is presented in Table 5.8. The coefficients of determination (*i.e.* shared variance (vs)) were computed using eq. 5.4.

$$r = \frac{n\sum(p \times q) - (\sum(p) \times \sum(q))}{\sqrt{[n\sum p^2 - (\sum p)^2][n\sum q^2 - (\sum q)^2]}} \tag{5.3}$$

In eq. 5.3, 'n' represents the sample size and p, q are the studied variables.

$$vs = r^2 \times 100 \tag{5.4}$$

Considering the static execution models, The 'r' values and VS were computed for each

Table 5.8: Correlation Intervals and Level of Significance.

Min	Max	Significance	Direction
-1.00	-0.50	High	Negative
-0.49	-0.30	Medium	Negative
-0.29	-0.10	Small	Negative
-0.09	+0.09	No	-
0.10	0.29	Small	Positive
0.30	0.49	Medium	Positive
0.50	1.00	High	Positive

Table 5.9: Correlation Coefficients and Shared Variance for Static Execution Models.

		RAM,Power	RAM,Makespan	Power,Makespan
Far-edge Devices	r (App1)	0.318	-0.056	-0.023
	VS (App1)	10.11%	0.31%	0.05%
	r (App2)	-0.032	0.1	0.101
	VS (App2)	0.10%	1%	1.02%
	r (App3)	0.469	0.094	0.483
	VS (App3)	21.99%	0.88%	23.32%
F2F	r (App1)	0.07	-0.006	-0.109
	VS (App1)	0.49%	0.003%	1.19%
	r (App2)	-0.075	0.019	-0.147
	VS (App2)	0.56%	0.036%	14.71%
	r (App3)	0.274	0.098	-0.102
	VS (App3)	7.51%	0.96%	1.04%
F2C	r (App1)	-0.585	-0.02	-0.017
	VS (App1)	34.22%	0.04%	0.029%
	r (App2)	0.042	-0.012	-0.071
	VS (App2)	0.18%	0.014%	0.50%
	r (App3)	-0.666	-0.109	-0.044
	VS (App3)	44.35%	1.19%	0.19%

MDSM applications (*i.e.* App1, App2, and App3) and presented in Table 5.9. Following hypothesis were constructed for correlation analysis whereby H_0 presents the null hypothesis and H_1 represents the alternate hypothesis.

Correlation Analysis for Far-edge Mobile Devices

H_0 : For App1, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App1, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: Referring Table 5.9, a positive correlation (*i.e.* r=0.318) between memory utilization and battery power consumption was witnessed whereby 10.11% of the executions share the variances. However, the correlation of makespan with both memory and battery power consumption remained negative and insignificant whereby the shared variance remained below 0.32%. Conclusively, H_1 is accepted for memory and battery power relationship however H_0 is accepted for correlations of makespan with memory and battery power consumption.

H_0 : For App2, there is no significant correlation between batter power consumption, memory utilization, and makespan.

H_1 : For App2, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: The analysis shows that there is insignificant correlation among memory utilization, battery power consumption, and makespan. It was found that memory utilization and power consumption are negatively correlated however the relationship is not significant (*i.e.* r=-0.032). Alternatively, makespan is positively related with memory utilization and power consumption but the relationship is minutely significant in this case. Considering these correlation, H_0 is accepted for all three relationships.

H_0 : For App3, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App3, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: The 'r' value for correlation among makespan and memory utilization was found as 0.094 which results in 0.88% variance among variables. Therefore the relationship remained insignificant. However, strong correlations of power consumption with makespan and memory utilization were observed whereas the 'r' values remained 0.469 and 0.483, respectively. The VS for power and memory remained 21.99% however

it increased up to 23.32% in the case of relationship between power consumption and makespan. Considering these values, H_0 is accepted for the correlation between memory consumption and makespan and H_1 is accepted for rest of the correlations.

Correlation Analysis for F2F Communication Model

H_0 : For App1, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App1, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: The 'r' values show the positive correlation between memory utilization and power consumption, however, the correlation is not significant enough to distinguish the relationship among both variables. Alternatively, the correlation of makespan with power consumption and memory utilization is negatively directed but this correlation is also not very significant. The VS values of variables fluctuate between 0.003% and 1.19%. Considering these results, H_0 is accepted for all three variables.

H_0 : For App2, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App2, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: The 'r' values show the negative correlation of battery power consumption with makespan and memory utilization. However, the correlation between battery power consumption and memory utilization is not significant enough to be considered. The correlation of battery power consumption with makespan has medium level significance with VS values as 14.71%. Alternatively, makespan and memory utilization are positively correlated but there is no significant correlation observed between the variables. Considering these correlations, H_0 is accepted for the correlations of memory utilization with battery power consumption and makespan. However, for the correlation

between makespan and battery power consumption, H_1 is accepted.

H_0 : For App3, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App3, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: The analysis show that memory utilization is positively correlated with other variables. However, the correlation between battery power consumption and makespan is negatively directed. The medium level significance was observed for the correlation between memory utilization and battery power consumption. Similarly, medium but not very much significant correlation was observed between battery power utilization and makespan whereas the VS remained 1.04%. On the other hand, no significant correlation was observed between memory utilization and makespan. Considering these results, H_0 is accepted for the relationship between memory utilization and makespan. For rest of the correlations, H_1 is accepted.

Correlation Analysis for F2C Communication Model

H_0 : For App1, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App1, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: The 'r' values show the negative correlation among all variables. However, memory utilization and battery power consumption are highly correlated (*i.e.* r= -0.585) having VS as 34.22%. The relationships of makespan with battery power consumption and memory utilization were not significant enough. Considering these results, $H1$ is accepted for the relationship between memory utilization and battery power consumption. For other relationships, H_0 is accepted.

H_0 : For App2, there is no significant correlation between battery power consump-

tion, memory utilization, and makespan.

H_1 : For App2, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: Referring Table 5.9, the correlation of makespan with other variables is negatively directed and the correlation coefficient for memory utilization and power consumption has positive direction. However, the 'r' values shows the insignificant correlation among all variables under consideration and the VS values fluctuate between 0.014% and 0.50%. Therefore, H_0 is accepted for all correlations.

H_0 : For App3, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App3, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: The 'r' values show that all correlation coefficients are negatively directed. The correlation between memory utilization and battery power consumption is highly significant (*i.e.* r=-0.666) having VS of 44.35%. On the other hand, the correlation between memory utilization and makespan has medium level of significance having VS as 1.19%. However, the correlation between power consumption and makespan is not significant. Considering these results, H_0 is accepted for the correlation between battery power consumption and makespan. For rest of the correlations, H_1 is accepted.

Overall, the correlation analysis show that there is a medium level of significance between memory utilization and battery power consumption when MDSM applications were executed using far-edge mobile devices and F2F communication models. However, there exists a high but negative correlation between memory utilization and power consumption when applications were executed using F2C communication model. On the other hand, the memory utilization and makespan has no correlation in all three execution models. Finally, the correlation between battery power consumption and makespan has

medium level of significance.

This section presented performance evaluation of static execution models of MDSM applications in far-edge mobile devices, and F2F and F2C communication models. Since there is no simulator available for the evaluation of existing systems, therefore these systems are mapped onto static execution models in order to compare with the performance of proposed dynamic and adaptive execution models (see Figure 5.24) later in this chapter.

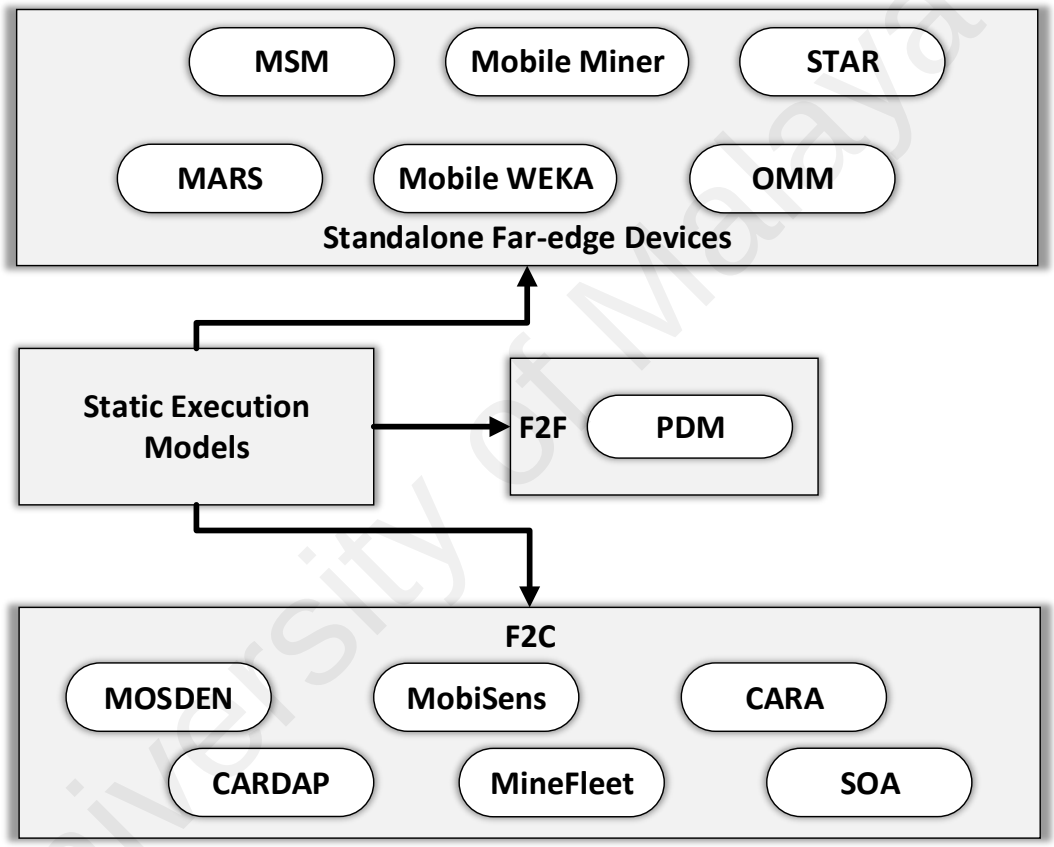


Figure 5.24: Existing Systems and their Mappings with Static Execution Models.

5.3.4 Dynamic Application Execution using UniMiner

The proposed dynamic execution model was evaluated by enabling all three communication models (*i.e.* standalone, F2F, and F2C) at the same time in UniMiner architecture.

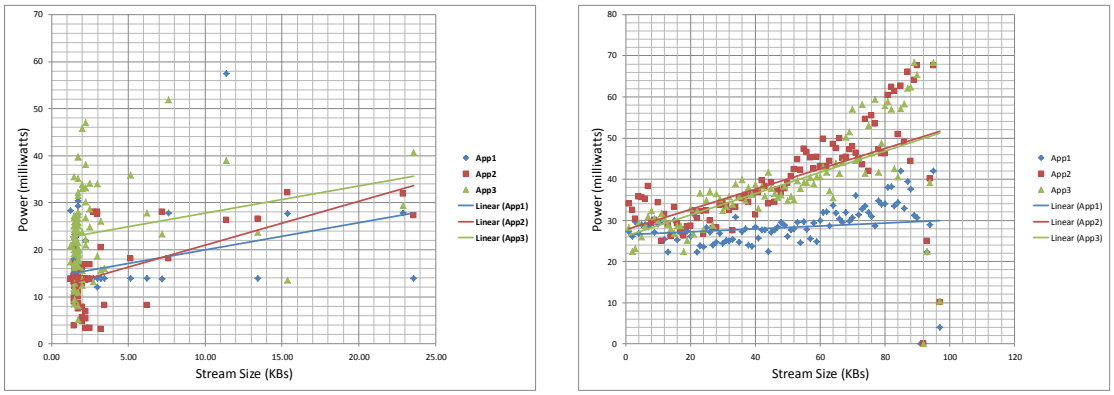
Experimental Procedure: The experiments were performed using four mobile devices (*i.e.* D1, D2, D4, and D5) by enabling the Wi-Fi interfaces. The devices D1 and D2 function as client devices whereas the devices D4 and D5 work as mobile edge

servers. Three versions of each MDSM application were developed to run in client mobile devices, mobile edge servers, and in CC servers. Since the main objective of dynamic execution model is to lower the makespan in MDSM applications by preferring local execution in client devices instead of mobile edge servers and CC servers. Therefore, client applications enable components for data acquisition, data adaptation, data preprocessing, data fusion, transient data stream management, device discovery, peer network formation, resource monitoring, context collection, adaptation engine, data offloading, data mining, knowledge integration, and knowledge management. The server application for mobile edge servers enabled components for data mining and pattern synchronization. Similarly cloud side application enabled the data mining and pattern synchronization services.

The experiments were performed by two users. It should be noted that the same learning models were reused which were developed in first phase of performance evaluation. First user put D1 in right pocket and D4 in left pocket and second user put D3 and D5 in right and left pockets of pants, respectively. The users were instructed to remain inside university premises in order to asses the performance of execution model within same Wi-Fi network. The mobility factors in switching to other networks were ignored in this experiment. The experiment were run for nine hours in total whereby each MDSM application was executed for three hours. It should be noted that number of events stored in each data file during transient data stream management varied randomly whereas the maximum events stored in a single file were set as 1000.

Findings: The experiments were performed by setting the time interval for data rate as 10 ms. It was observed that the application did not crashed with the given data rate.

The power consumption trends on both devices varied and presented in Figure 5.25a and Figure 5.25b. The power consumption varies due to varying resource availability in both devices. It was observed that App3 on D1 consumed comparatively higher battery power while App1 and App2 consumed more or less same amount of power. Alterna-



(a) Power Consumption of D1.

(b) Power Consumption of D2.

Figure 5.25: Power Consumption using Dynamic Execution Model

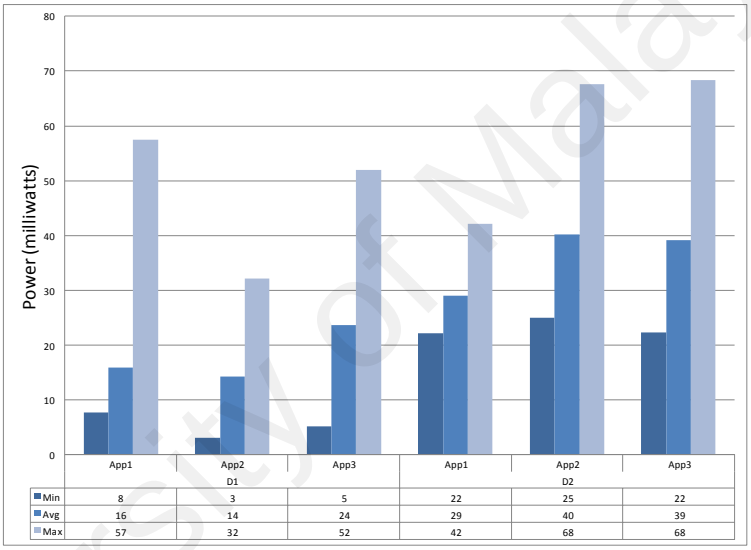


Figure 5.26: Power Consumption Trends of D1 and D2.

tively, using D2, App1 consumed comparatively lower power whereas App2 and App3 consumed more or less same amount of power. The overall power consumption trends of both devices are presented in Figure 5.26. To compare the performance of MDSM applications in both devices, the standard deviation was calculated using eq.5.2. It was observed that App1 has standard deviation of 6 mW on D1 which was lowered to 4 mW on D2. However, for App2, standard deviation is increased up to 10 mW on D2. For App3, the standard deviation on D1 was 9 mW which was slightly increased up to 10 mW on D2.

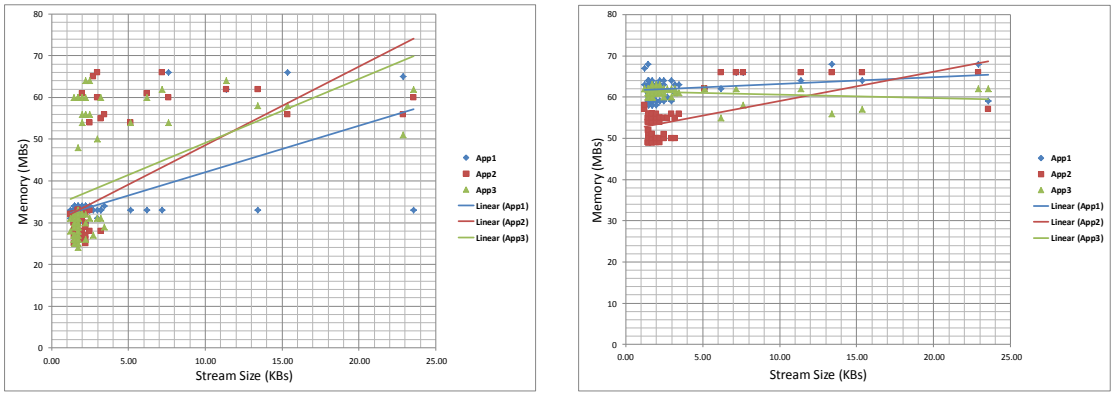
Table 5.10 presents the battery depletion time of both devices. It was found that

Table 5.10: Battery Charge Depletion Time using Dynamic Execution Model.

	D1					
	App1		App2		App3	
	P	$Time_{depletion}$	P	$Time_{depletion}$	P	$Time_{depletion}$
Min	8	55 Hrs 05 Mins	3	148 Hrs 02 Mins	5	89 Hrs 20 Mins
Avg-S.Dev	10	44 Hrs 42 Mins	8	55 Hrs 05 Mins	15	29 Hrs 36 Mins
Avg	16	28 Hrs 25 Mins	14	32 Hrs 11 Mins	24	18 Hrs 50 Mins
Avg+D.Dev	22	20 Hrs 18 Mins	20	22 Hrs 20 Mins	33	13 Hrs 45 Mins
Maximum	57	08 Hrs 19 Mins	32	14 Hrs 27 Mins	52	12 Hrs 33 Mins
	D2					
	App1		App2		App3	
	P	$Time_{depletion}$	P	$Time_{depletion}$	P	$Time_{depletion}$
Min	22	31 Hrs 28 Mins	25	27 Hrs 53 Mins	22	31 Hrs 28 Mins
Avg-S.Dev	25	27 Hrs 53 Mins	30	23 Hrs 34 Mins	29	24 Hrs 13 Mins
Avg	29	24 Hrs 13 Mins	40	17 Hrs 20 Mins	39	18 Hrs 04 Mins
Avg+D.Dev	33	21 Hrs 25 Mins	50	14 Hrs 16 Mins	49	14 Hrs 04 Mins
Maximum	42	16 Hrs 38 Mins	68	10 Hrs 12 Mins	68	10 Hrs 12 Mins

complete battery depletion time of D1 while running App1 varies between eight hours at minimum to 55 hours at maximum. However, battery depletion time of D2 for App1 fluctuates between 16 hours and 31 hours. Similar trends were uncovered for App2 and App3 whereby the battery depletion time has more variation on D1 as compared to D2. Overall, the battery depletion time of D1 varies between 08 hours at minimum to 148 hours at maximum. For D2, it varies between minimum 10 hours to maximum 31 hours.

The memory consumption trends of both devices are presented in Figures 5.27 and Figure 5.28. It was observed that, on average, D1 consumed lower memory (*i.e.* between 34 MB and 38 MB) because of low memory availability. Alternatively, the average memory consumption on D2 remained between 54 MB and 62 MB. It was also observed that during some executions, the memory consumption at D1 drastically increased almost doubling the average. Therefore, the standard deviations results of D1 remained comparatively higher. The standard deviations on D1 were found as 6.72 MB, 11.7 MB, and 13.65 MB for App1, App2, and App3, respectively. However, the memory consumption on D2 remained comparatively stable whereas the standard deviations for App1, App2, and App3 remained 2.38 MB, 4.44 MB, 1.3 MB, respectively. Considering the memory



(a) Memory Consumption of D1. (b) Memory Consumption of D2.

Figure 5.27: Memory Consumption using Dynamic Execution Model

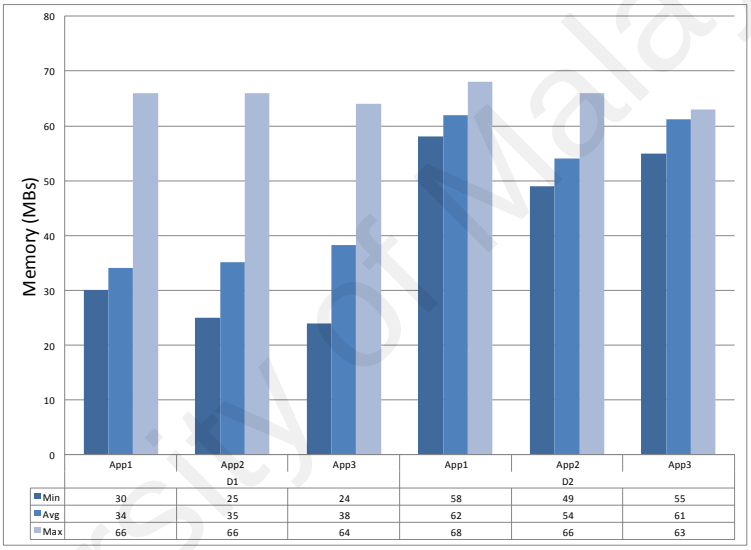
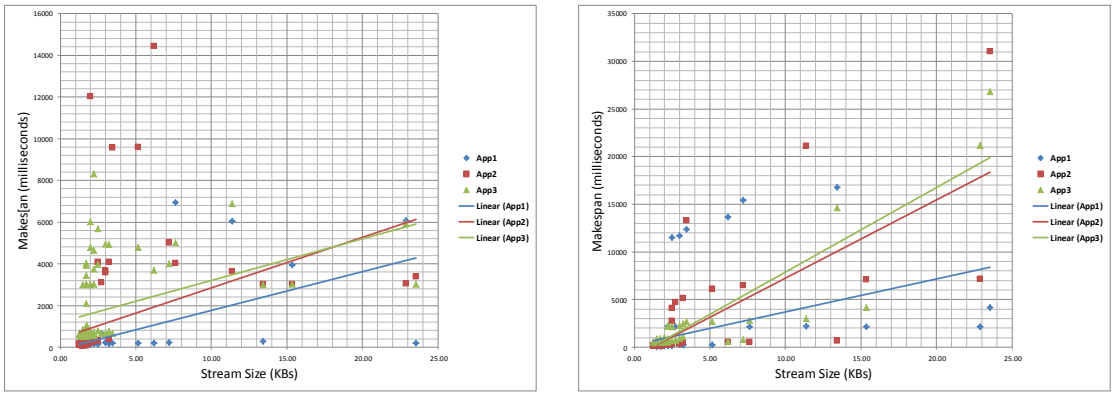


Figure 5.28: Memory Consumption Trends of D1 and D2.

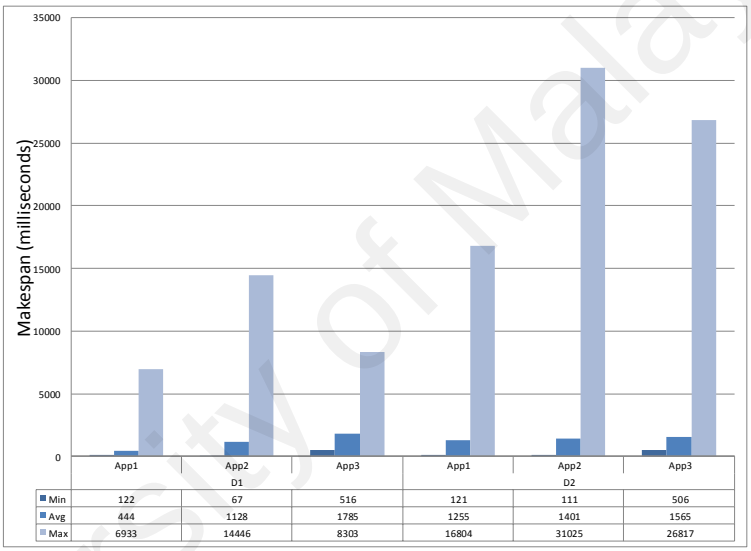
consumption at application level, App3 consumed comparatively higher memory at D1, whereas there is no significant difference between App1 and App3 observed. On the other hand, at D2, App2 consumed comparatively lower memory, whereas an insignificant difference in memory consumption was observed for App1 and App3. At device level, D2 consumed comparatively higher memory because of more memory availability. However, the average memory consumption on D1 accounts for about 4.32% of total available memory (*i.e.* 832 MB) which is decreased up to 2.12% of total available memory (*i.e.* 2783 MB) in D2. Therefore, it was found that dynamic application execution does not hampers the memory resources of mobile devices.



(a) Makespan of D1.

(b) Makespan of D2.

Figure 5.29: Makespan using Dynamic Execution Model



	App1	App2	App3	App1	App2	App3
	D1			D2		
Min	122	67	516	121	111	506
Avg	444	1128	1785	1255	1401	1565
Max	6933	14446	8303	16804	31025	26817

Figure 5.30: Makespan Trends of D1 and D2.

Figure 5.29 and Figure 5.30 present the makespan variations and trends during dynamic application execution. Since the execution model enables to dynamically switch between three different modes of execution. Therefore, significant variations in makespan values were observed. For example the makespan values for App1 on D1 fluctuated between 122 ms and 6933 ms and on D2 the makespan fluctuated between 122 ms and 16804 ms. Similar trends were observed for App2 and App3 on both devices. Overall, the average makespan on D1 remained 444 ms, 1128 ms, and 1785 ms for App1, App2 and App3. Alternatively, the average makespan on D2 remained 1255 ms, 1401 ms, and 1565 ms for App1, App2, and App3, respectively. Considering the variations in makespan, the com-

puted standard deviation was 1169 ms, 2556 ms, and 1825 ms for App1, App2, and App3 for D1. It was found that standard deviation for App1 on D1 is about 62% higher as compared with average makespan. Similarly, for App2, the difference between average and standard deviation of makespan increases about 56% of average makespan. For App3, this difference reduces to about 2%. On the other hand, the average makespan on D2 remained 1255 ms, 1401 ms, and 1565 ms for App1, App2, and App3 respectively. However, the standard deviation remained 63%, 67%, and 41% of average makespan for App1, App2, App3, respectively. The variations in statistics show that neither the available resources in far-edge mobile devices nor the size of data stream impacts the makespan. However communication relevant issues such as connecting ad hoc mobile edge servers and transmitting data to cloud servers from different bandwidth connections increase the makespan in MDSM applications. Considering overall average makespan, MDSM applications on D1 performed comparatively better. Apparently, it looks that dynamic execution model performs worse in terms of makespan; when considered the amount of data processed in all three modes, it was found that most of the executions were performed inside far-edge

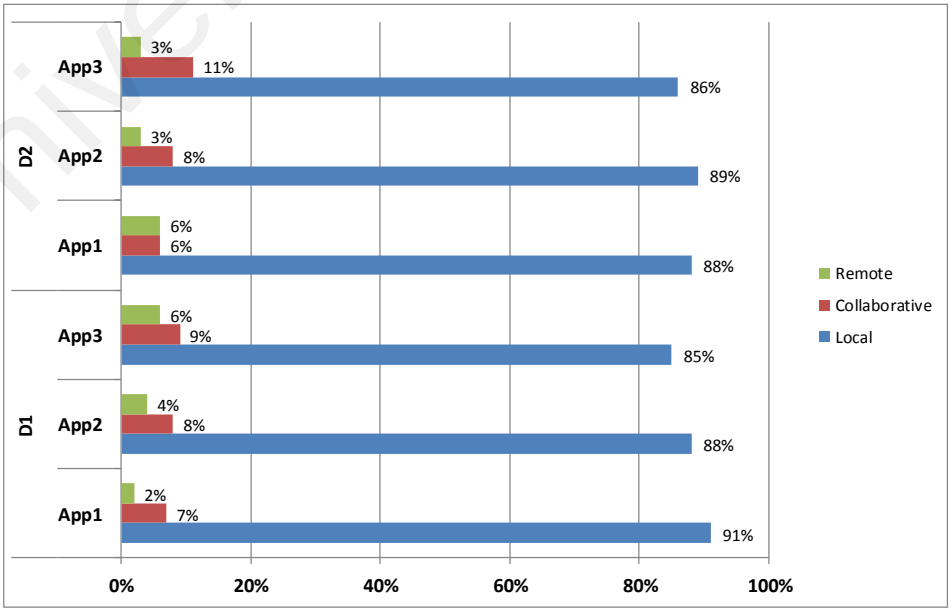


Figure 5.31: Data Reduction using Dynamic Execution Model.

devices whereas the execution model occasionally switched to other execution modes. Figure 5.31 shows the amount of data processed in each mode. It was observed that each MDSM application processed at least 85% of data using far-edge devices whereas at most 11% of data was processed using mobile servers and 6% by CC servers. These statistics shows that adoption of dynamic execution model is favorable for MDSM application execution because of reduced dependency over mobile edge servers and CC servers.

Considering the performance of MDSM applications, the accuracy trends are presented in Figure 5.32. The results show insignificant variations however the applications on D1 produced comparatively lower accuracy. The produced accuracy remained as low as 78.9% and as high as 79.7%. The accuracy varied because the learning models were reused however the testing was performed in real settings by different users.

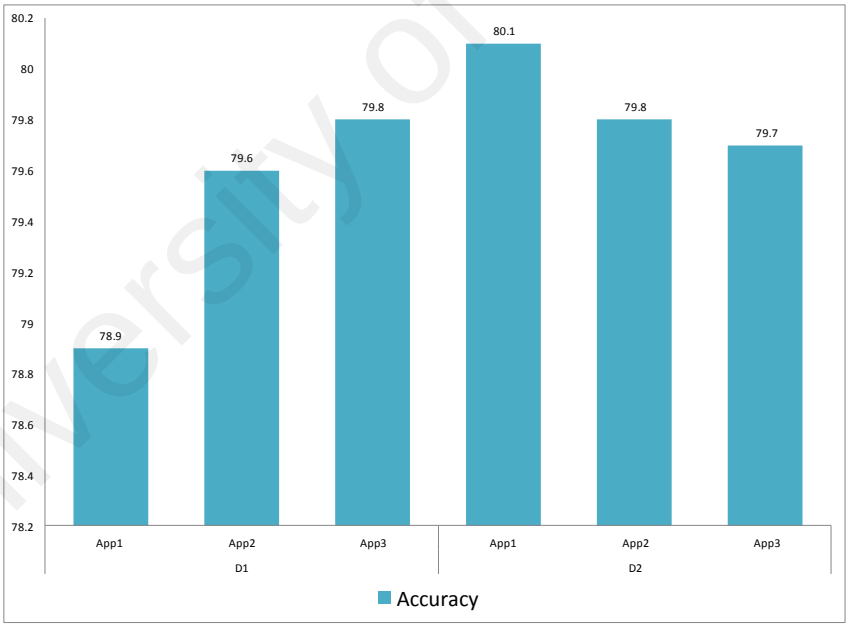


Figure 5.32: Accuracy Analysis using Dynamic Execution Model.

Correlation analysis for dynamic execution model

H_0 : For App1, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App1, there is a significant correlation between battery power consumption,

Table 5.11: Correlation Coefficients and Shared Variance in Dynamic Execution model.

	RAM,Power	**RAM,Makespan**	**Power,Makespan**
r (App1-D1)	0.583	0.959	0.677
VS (App1-D1)	33.99%	91.97%	45.83%
r (App2-D1)	0.722	0.778	0.275
VS (App2-D1)	52.12%	60.52%	7.56%
r (App3-D1)	0.647	0.887	0.620
VS (App3-D1)	41.86%	78.68%	38.44%
r (App1-D2)	0.182	0.245	0.503
VS (App1-D2)	3.31%	6%	25.3%
r (App2-D2)	0.455	0.378	0.479
VS (App2-D2)	20.7%	14.29%	22.94%
r (App3-D2)	-0.179	-0.109	0.589
VS (App3-D2)	3.20%	1.19%	34.69%

memory utilization, and makespan.

Significance: The 'r' values relevant to App1 show the significant correlation among all three variables for both devices (*i.e.* D1 and D2) (see Table 5.11). For D1, the high level significance among all variables having positive correlations was witnessed. However, for D2, the correlation of memory utilization with battery power consumption and makespan remained medium but high level of significance was observed for the correlation between battery power consumption and makespan. The VS in D1 varies between 33.99% and 91.97% but on D2 it lowers and remained between 3.31% and 25.3%. Considering these results, H_1 is accepted for relationship among all variables.

H_0 : For App2, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App2, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: The 'r' values for App2 show the positive correlation among all variables in both devices. For D1, the correlation of memory utilization with other variables remained significantly high however small level of significance was witnessed for correlation between battery power consumption and makespan. For D2, the medium level of

correlation among all variables was witnessed. The VS on D1 varies between 7.56% and 60.52% and it varies between 14.29% and 22.94% on D2. Considering these results, H_1 is accepted for all correlations.

H_0 : For App3, there is no significance correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App3, there is a significance correlation between battery power consumption, memory utilization, and makespan.

Significance: The 'r' values show that all variables on D1 have high level of positive correlation. On D2, the correlation between battery power consumption and makespan is high and positively directed but the correlation of memory consumption with battery power consumption and makespan remained small and negatively directed. The VS on D1 remained between 38.44% and 78.68% however it fluctuates between 1.19% and 34.69% on D2. Considering these variations, the H_1 is accepted for all variables on D1 but, for D2, H_1 is accepted for the correlation between battery power consumption and makespan, H_0 is accepted for other correlations.

The overall analysis show a positively significant correlation among makespan, memory utilization, and battery power consumption for dynamic execution model.

5.3.5 Adaptive Application Execution using UniMiner

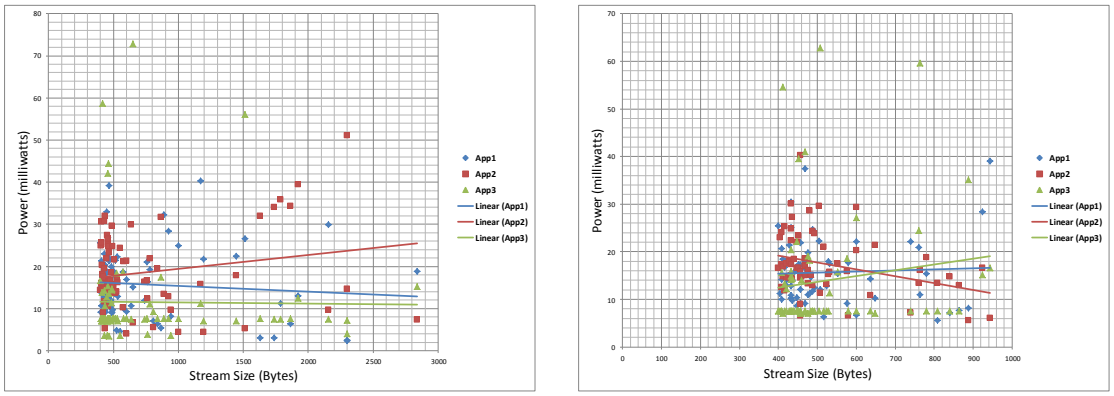
The enforced execution of MDSM application components in mobile devices and the manual setting of file size challenges the seamless application execution in UniMiner. Therefore the experiments were performed to assess the performance of adaptive application execution models.

Experimental Procedure: The experiments were performed using four mobile devices (*i.e.* D1, D2, D4, and D5) by two users moving inside university's Wi-Fi network. In these experiments, three versions of each MDSM application were developed to run in

client far-edge devices, mobile edge servers, and CC servers. The client side application enabled application components for data acquisition, data adaptation, data preprocessing, data fusion, data mining, knowledge integration, and knowledge management. In addition resource monitoring, context collection, and adaptation engine components were enabled to deploy proposed multiple point data stream management and multiple point data offloading schemes. The mobile edge server application enabled the components for data synchronization, data preprocessing, data fusion, data mining, and pattern synchronization. Similarly, the cloud application enabled cloud services for data synchronization, data preprocessing, data fusion, data mining, and pattern synchronization. It should be noted that the same learning models were reused which were developed in first phase of performance evaluation. The experiments were run for nine hours whereas each MDSM application were run for three hours. During experiments, first user put D1 in right pocket and D4 in left pocket and second user put D2 in right pocket and D5 in left pocket.

Findings: The performance evaluation of MDSM applications were made for all application components. An interesting phenomenon was observed that the use-case application performed transition between execution modes from only two points. The applications performed switching either after raw data collection and data adaptation or the applications performed in-memory operations for data preprocessing and data fusion using onboard resources in far-edge devices and switched to other modes in order to perform data mining and knowledge management operations. Therefore, the results presented in this section shows performance of MDSM applications from two points of execution i.e. P_1 and P_2 whereas P_1 represents first switching point and P_2 is the second point for application switching.

The battery power consumption results of MDSM applications are presented in Figure 5.33 and Figure 5.34. The results show that most of the data streams were processed with small size and power consumption varied significantly. It was observed that, on av-



(a) Power Consumption at P_1.

(b) Power Consumption at P_2.

Figure 5.33: Power Consumption using Adaptive Execution Model

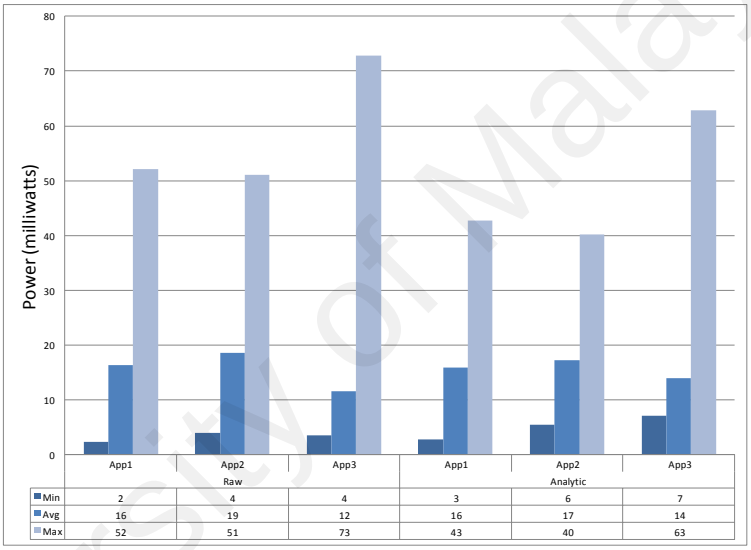


Figure 5.34: Power Consumption Trends at P_1 and P_2.

erage, App3 consumed comparatively lower power *i.e.* 10 mW at P_1 and 14 mW at P_2. It was also observed that difference between minimum and maximum power consumption widened as compared to other applications (see Figure 5.34). The standard deviation in power consumption at P_1 remained 9 mW for App1 and App2, and 12 mW for App3. Similarly, at P_2, the standard deviation remained 8 mW, 6 mW, and 12 mW for App1, App2, and App3, respectively. Considering variations in power consumption, the battery depletion time for MDSM applications is presented in Table 5.12. The average battery depletion time at P_1 varies between 12 hours and 222 hours for App1, between 09 to 111 hours for App2, and it remained between 6 hours 111 hours for App3. Similarly, for P_2,

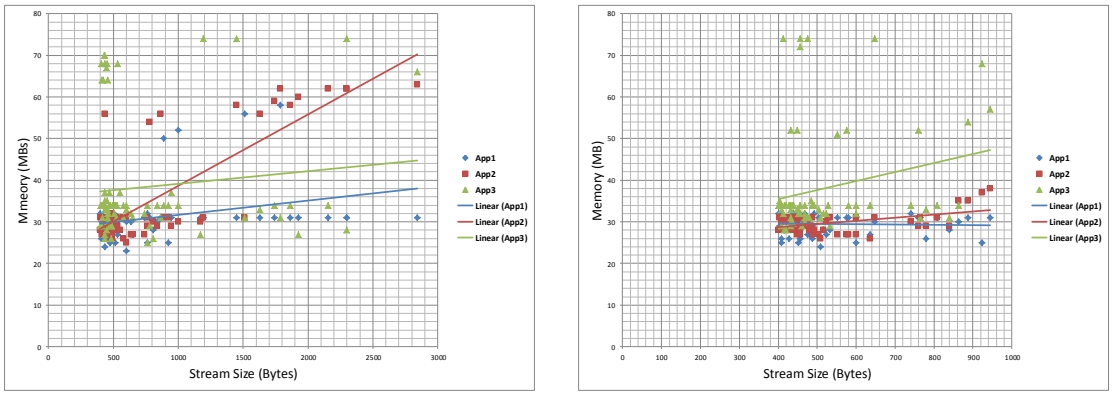
Table 5.12: Battery Charge Depletion Time using Adaptive Execution Model.

	P_1					
	App1		App2		App3	
	P	$Time_{depletion}$	P	$Time_{depletion}$	P	$Time_{depletion}$
Min	2	222 Hrs 05 Mins	4	111 Hrs 08 Mins	4	111 Hrs 01 Mins
Avg-S.Dev	7	63 Hrs 43 Mins	10	44 Hrs 40 Mins	3	148 Hrs 01 Mins
Avg	16	28 Hrs 25 Mins	19	23 Hrs 36 Mins	12	37 Hrs 01 Mins
Avg+D.Dev	25	18 Hrs 16 Mins	28	16 Hrs 36 Mins	21	21 Hrs 14 Mins
Maximum	52	12 Hrs 33 Mins	51	09 Hrs 11 Mins	73	06 Hrs 08 Mins
	P_2					
	App1		App2		App3	
	P	$Time_{depletion}$	P	$Time_{depletion}$	P	$Time_{depletion}$
Min	3	229 Hrs 40 Mins	6	115 Hrs 10 Mins	7	98 Hrs 31 Mins
Avg-S.Dev	8	86 Hrs 02 Mins	11	62 Hrs 56 Mins	2	344 Hrs 12 Mins
Avg	16	43 Hrs 01 Mins	17	40 Hrs 48 Mins	14	49 Hrs 15 Mins
Avg+D.Dev	24	29 Hrs 18 Mins	23	30 Hrs 32 Mins	26	26 Hrs 46 Mins
Maximum	43	16 Hrs 01 Mins	40	17 Hrs 20 Mins	63	11 Hrs 32 Mins

the average battery depletion time remained between 16 hours and 229 hours for App1, between 17 hours and 115 hours for App2, and between the interval of 11 hours and 344 hours for App3.

The memory consumption of MDSM applications vary at both P_1 and P_2 (see Figure 5.35). It was observed that MDSM applications consumed relatively more memory at P_1. In addition, the applications performed raw data offloading when stream size grows more than 500 Bytes. Although applications showed variations however memory consumption linearly grows for App1 and App3 at both points. However, for App2, the memory consumption significantly grows at P_1 but linearly on P_2. Considering the standard deviation, the memory consumption at P_1 varies as 5.84 MB for App1, 5.66 MB for App2, and 12.84 MB for App3. At P_2, the memory consumption varied about 2.23 MB for App1, 2.27 MB for App2, and about 12.69 MB for App3. Despite variations, the MDSM applications do not impact the overall performance of far-edge devices. On average, mobile devices utilized 29 MB to 40 MB which accounts between 3.48% to 4.80% of available memory at D1 and 1.04% and 1.44% of available memory and D2.

The makespan in MDSM applications using adaptive execution model were mea-



(a) Memory Consumption of P_1.

(b) Memory Consumption of P_2.

Figure 5.35: Memory Consumption using Adaptive Execution Model

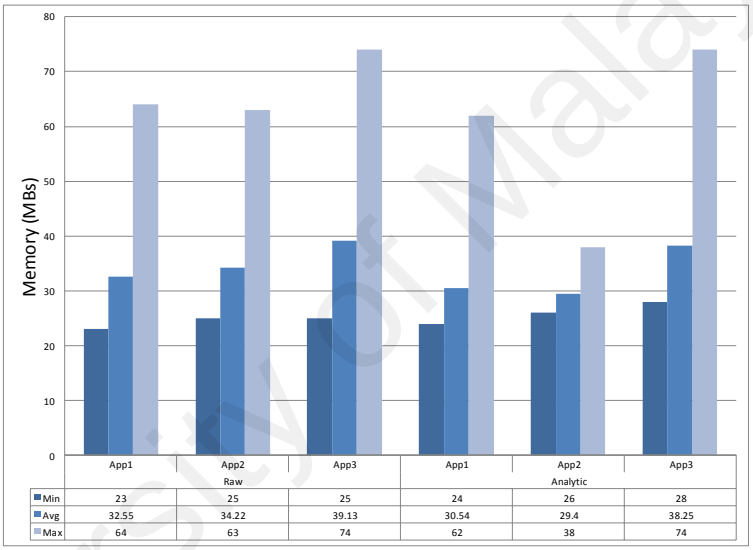
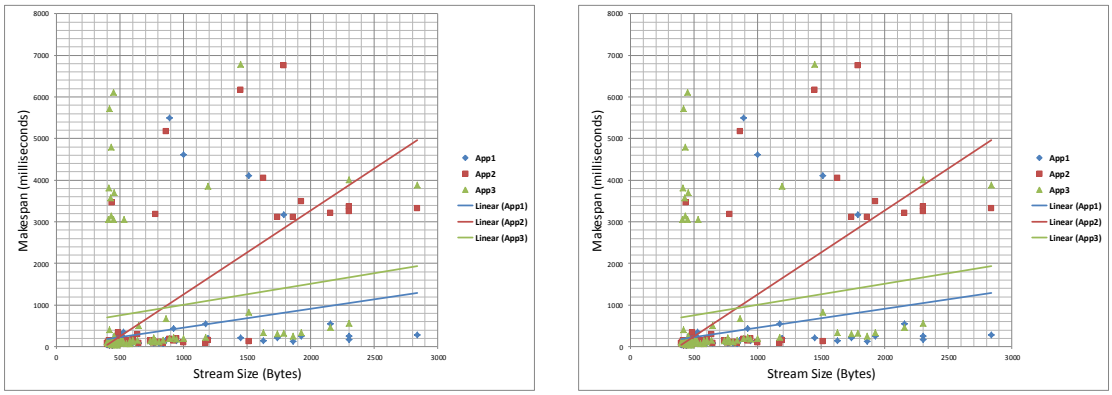


Figure 5.36: Memory Consumption Trends at P_1 and P_2.

sured similarly as it was measured in the case of dynamic execution model. However, the makespan values were measured from multiple switching points (see Figure 5.37 and Figure 5.38). It was observed that makespan values has variations as low as 33 ms and as high as 6792 ms. However, at P_1, the MDSM applications have comparatively higher makespan. On average, App1 had lowest makespan at P_1, followed by App2 and App3. Alternatively, at P_2, App2 had the lowest makespan, followed by App1 and App3. Considering the variation in makespan, the standard deviation at P_1 remained 1286 ms for App1, 1504 ms for App2, and 1746 ms for App3. On P_2, the standard deviation remained 449 ms, 60 ms, 1174 ms. It was observed that with some applications the standard devi-



(a) Makespan of P_1. (b) Makespan of P_2.

Figure 5.37: Makespan using Adaptive Execution Model

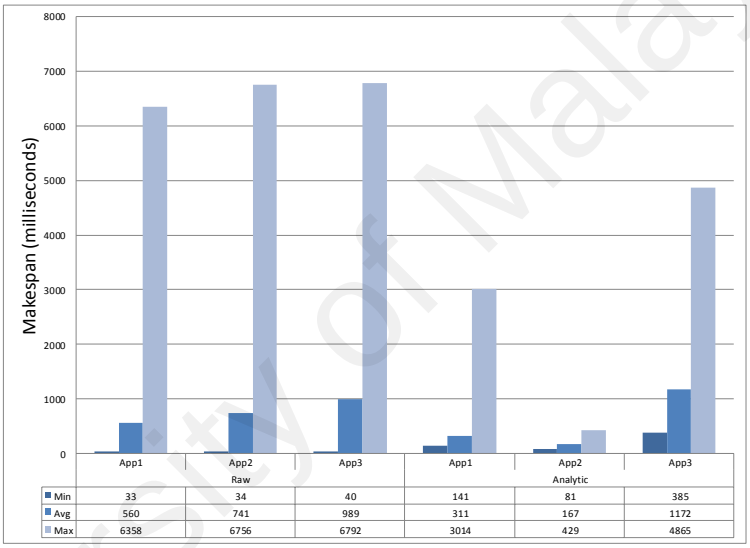


Figure 5.38: Makespan Trends at P_1 and P_2.

ation supersedes the double of average values which shows that few executions produced higher makespan as compared to other executions. However, overall makespan at P_1 remained 1512 ms and at P_2 it decreased to 561 ms.

The data reduction trends for adaptive execution model are presented in Figure 5.39. It was observed that most of the data streams were processed inside far-edge devices. For example, for App1 51% of the data stream was fully processed inside far-edge devices which becomes the best case for application execution. However, 26% of the data stream was immediately offloaded from P_1 which becomes the worst case scenario for application execution. Considering the overall data reduction results, about 21.33% of the data stream

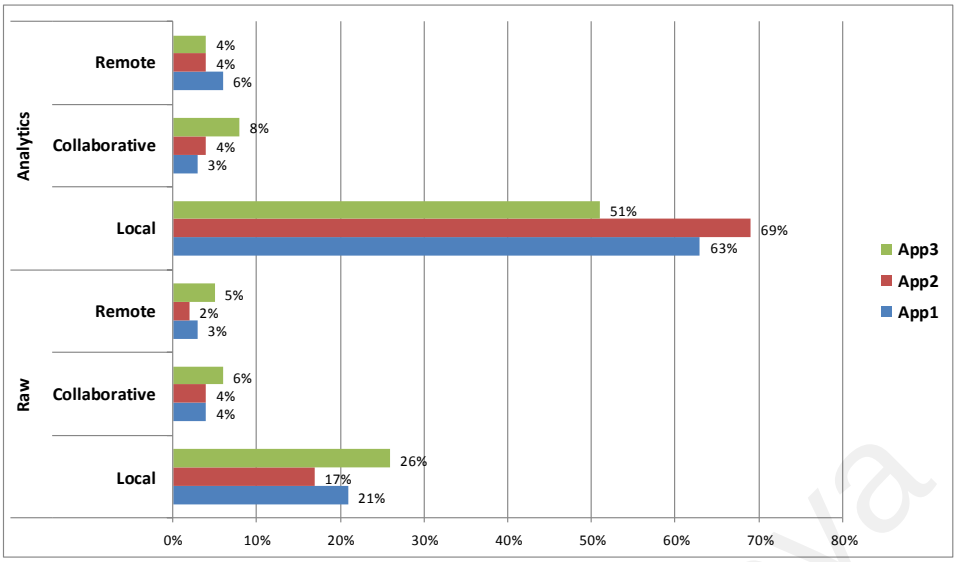


Figure 5.39: Data Reduction using Adaptive Execution Model.

were directly offloaded from P_1 whereby 58.32% was processed by mobile edge servers and 41.68% was processed by CC servers. Alternatively, from P_2, on average 9.68% of the data stream was offloaded whereby 51.75% was processed by mobile edge servers and 48.24% was processed by CC servers. The data reduction analysis shows that proposed execution model reduces 82.33% of data stream using far-edge devices, and 91.66% of the data streams using far-edge devices and mobile edge servers. It was observed that only 8.34% of the data stream was offloaded to CC servers.

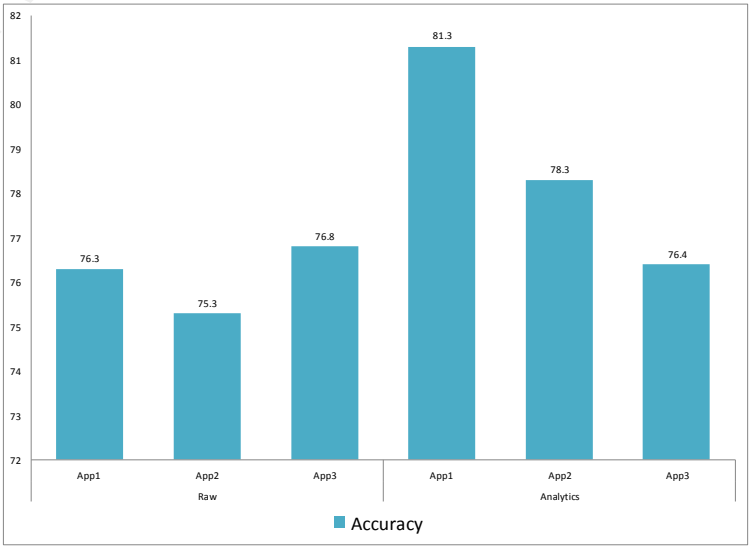


Figure 5.40: Accuracy Analysis using Adaptive Execution Model.

Figure 5.40 shows the accuracy trends of MDSM applications using adaptive execution model. It was observed that MDSM applications produced comparatively higher accuracy at P_2 which ranges between 76.4% to 81.3%. However due to raw data transfer and possible induction of noise and missing values in the data streams, the MDSM applications produced lower accuracy which ranges from 75.3% to 76.8%. Since the accuracy of MDSM applications depends upon the quality of learning models therefore the problem of low accuracy could be handled easily by training good learning models.

Correlation analysis for adaptive execution model

Table 5.13 presents the correlation coefficients and shared variance among studied variables. Following hypotheses were constructed for correlation analysis.

H_0 : For App1, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App1, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: The 'r' values show the positive correlation among all variables at P_1. At P_2, the memory consumption is negatively correlated with makespan and battery power consumption but the correlation between makespan and battery power consumption is positively directed. At P_1, the level of significance for correlation between memory utilization and battery power consumption is small, which increased to medium for correlation between power consumption and makespan and high between memory consumption and makespan. At P_2, there is no significance in correlation of memory consumption and makespan. However, medium level of correlation was witnessed in the relationships of makespan with memory utilization and battery power consumption. The VS at P_1 varies between 3.13% and 51.12%. Alternatively, at P_2, the VS varies between 0.65% and 11.56%. Considering these results, H_1 is accepted for all variables at P_1. At P_2, the H_0 is accepted for correlation between memory utilization and battery power con-

Table 5.13: Correlation Coefficients and Shared Variance of Adaptive Execution Model.

	RAM,Power	**RAM,Makespan**	**Power,Makespan**
r (App1-P_1)	0.177	0.715	0.337
VS (App1-P_1)	3.13%	51.12%	11.36%
r (App2-P_1)	0.310	0.928	0.378
VS (App2-P_1)	9.61%	86.11%	14.28%
r (App3-P_1)	-0.205	0.925	-0.110
VS (App3-P_1)	4.20%	85.56%	1.21%
r (App1-P_2)	-0.081	-0.340	0.316
VS (App1-P_2)	0.65%	11.56%	9.99%
r (App2-P_2)	-0.376	-0.218	0.067
VS (App2-P_2)	14.13%	4.75%	0.45%
r (App3-P_2)	-0.068	0.947	0.091
VS (App3-P_2)	0.46%	89.68%	0.83%

sumption. Alternatively, H_1 is accepted for correlation of makespan with battery power consumption and memory utilization.

H_0 : For App2, there is no significant correlation between battery power consumption, memory utilization, and makespan.

H_1 : For App2, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: For App2, similar trends were observed whereby at P_1 there is a positive correlation among all variables and at P_2 both the positive and negative correlations were observed among variables. At P_1, medium and high levels of significance were observed however, at P_2, correlation of memory utilization with battery power consumption and makespan was negative. The correlation between memory utilization and battery power consumption remained medium. Alternatively, a small level of significance was witnessed in correlation between memory utilization and makespan. However, there is no significance observed in the correlation between battery power consumption and makespan. The VS at P_1 varies between 9.16% and 86.11% and,at P_2, SV fluctuates between 0.45% and 14.13%. Considering these results, the H_1 is accepted for all correlations at P_1 and the correlation of memory utilization with battery power consumption

and makespan at P_2. However, H_0 is accepted for the correlation between battery power consumption and makespan.

H_0 : For App3, there is no significant correlation between battery power consumption, memory utilization, and makespan.

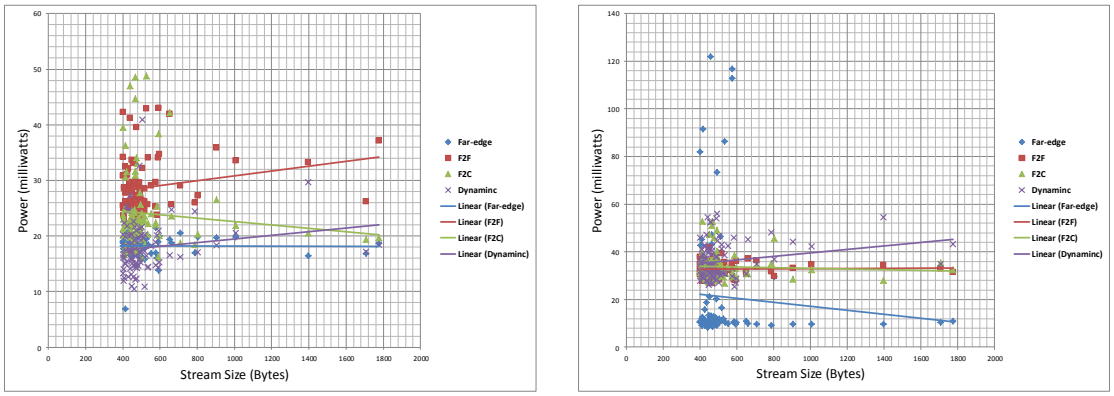
H_1 : For App3, there is a significant correlation between battery power consumption, memory utilization, and makespan.

Significance: At P_1, the 'r' values show the small level of significance in the correlations of battery power consumption with other variables. However high level of significance was observed in the correlation between memory utilization and makespan. Alternatively, at P_2, similar trends were observed for all variables. The VS at P_1 remained between 1.21% and 85.56% and, at P_2, VS fluctuates between 0.46% and 89.68%. Considering these results H_1 is accepted for the correlation between memory utilization and makespan at both P_1 and P_2. For rest of the correlations, H_0 is accepted.

The overall analysis shows the positive and significant correlation among all variables, however, at P_2, the correlation coefficients vary in positive and negative directions.

5.4 Discussion

Section 5.2 presented experimental evaluation of MDSM applications using static, dynamic, and adaptive execution models. Since the static models lack run-time executions in multiple modes. The dynamic execution models facilitate in addressing this issue but these models need human interventions during application deployment in order to set the maximum file size which is processable in far-edge devices. In addition, dynamic execution model bounds the execution of few components strictly in far-edge devices. The adaptive execution model facilitate in addressing these issues. The adaptive execution enables to switch application execution from multiple points of execution. In addition, it adapts the execution behavior from multiple points in MDSM applications. The static ex-



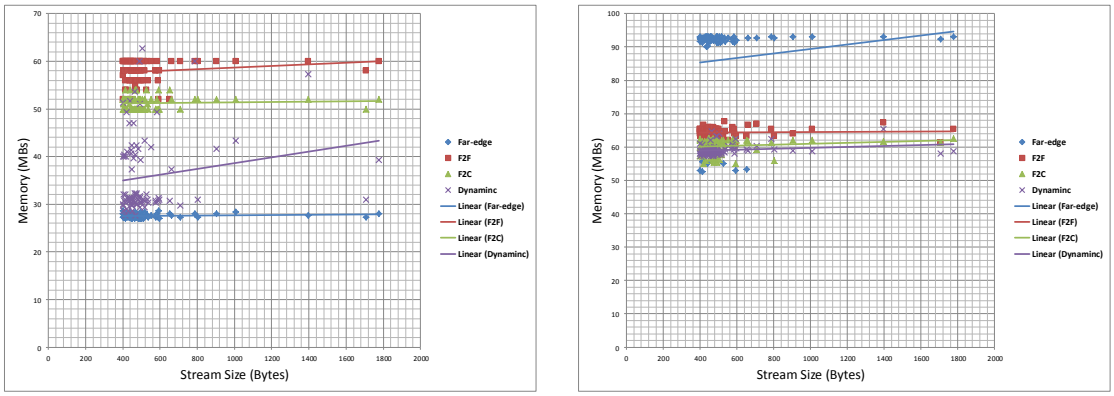
(a) Power Consumption Comparison of D1.

(b) Power Consumption Comparison of D2.

Figure 5.41: Power Consumption Comparisons - Static vs. Dynamic

ecution models are considered as benchmark for comparison with dynamic and adaptive execution models. It should be noted that static models replicate the application execution logic of several existing systems as shown previously in Figure 5.24.

The battery power consumption trends of static and dynamic execution models are presented in Figure 5.41. It was observed that on average static applications consumed 18 mW (using D1) and 29 mW (using D2) for each batch of sliding windows. Comparatively, dynamic applications consumed 18 mW (using D1) and 36 mW (using D2) on average for each batch of sliding windows. It was observed that dynamic execution model do not significantly impacts the extra battery power consumption because of size limitations of data stream. Since the model ensures to execute maximum data streams in far-edge devices therefore battery power consumption remains lower. It was also observed that in most of the executions, size of data streams was low therefore battery power consumption remained closer to static execution of data stream in far-edge devices. Comparatively, the average battery power consumption of static execution models using F2F settings remained 29 mw (using D1) and 33 mW (using D2). Similarly, for F2F communication model, the average battery power consumption remained 24 mW (using D1) and 34 mW (using D2). Although there exist some spikes in battery power consumption however the linear trend line shows that dynamic execution model performs comparatively better than



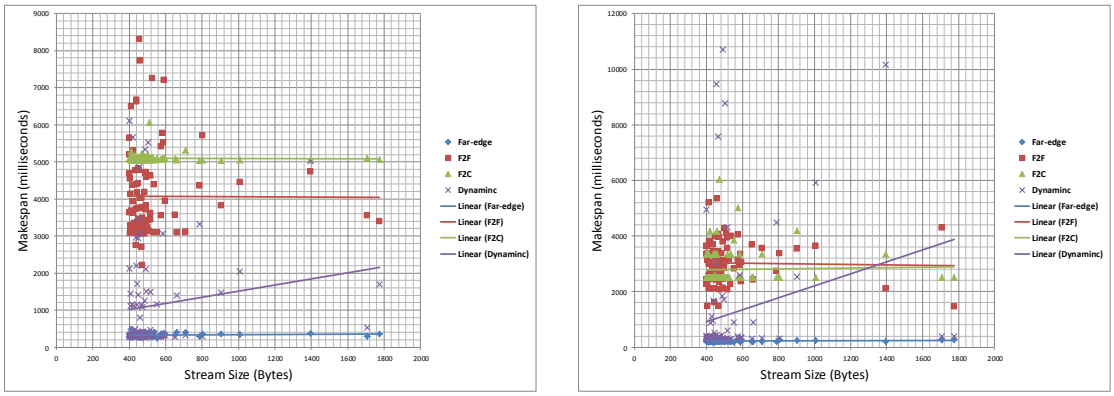
(a) Memory Consumption Comparison of D1. (b) Memory Consumption Comparison of D2.

Figure 5.42: Memory Comparisons - Static vs. Dynamic

static models which use F2F and F2C communication models but it can not outperform the static execution in far-edge mobile devices.

Figure 5.42 shows memory consumption of static and dynamic execution models using D1 and D2. It was observed that static execution in far-edge devices consumed 28 MB in D1 and 86 MB in D2. Alternatively, during dynamic executions, MDSM applications consumed 36 MB in D1 and 59 MB in D2. Comparatively static execution using F2F communication model incurred 58 MB and 64 MB on average and using F2C communication model, static applications consumed 51 MB in D1 and 60 MB in D2. The comparison shows that memory consumption of dynamic execution model remained higher than static application execution in far-edge mobile devices but it occasionally increased the memory utilization of static applications in F2F and F2C communication models.

As witnessed in section 5.2, the static application execution using F2F and F2C settings increases the makespan as compared with static execution in far-edge devices. Considering these facts, the dynamic model was designed to process maximum data stream using onboard computational resources. Therefore, the makespan in MDSM applications remained closer to the makespan of static applications in far-edge devices. Figure 5.43 shows the comparison of static and dynamic execution models. It was observed that on



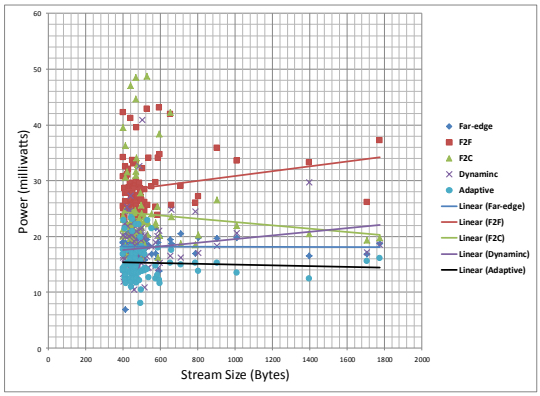
(a) Makespan Comparison of D1.

(b) Makespan Comparison of D2.

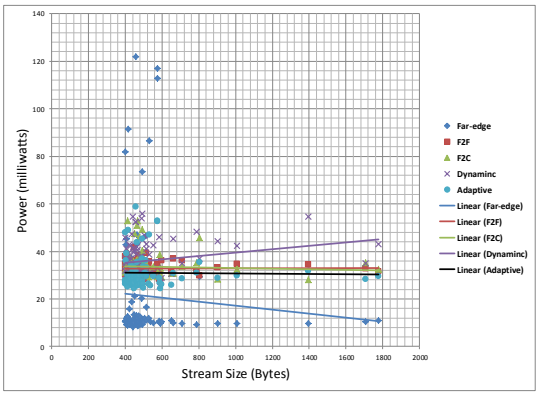
Figure 5.43: Makespan Comparisons - Static vs. Dynamic

average static applications in far-edge devices consumed 331 ms using D1 and 227 ms using D2. However, the makespan increases in static applications for F2F and F2C communication models. In F2F settings, MDSM applications consumed 4070 ms (using D1) and 3021 ms (using D2). Similarly, in F2C settings, the makespan was increased up to 5098 ms using D1 and 2787 ms using D2. However, the dynamic application execution brings 1119 ms of makespan at D1 and 1195 ms of makespan at D2 which is comparatively better than F2F and F2C models. This comparative evaluation shows that dynamic execution model outperforms the resource consumption using static execution models in F2F and F2C settings but the performance remains closer to the static application execution in far-edge devices.

The comparisons of proposed adaptive execution model with static and dynamic execution models show improved performance. Considering the battery power consumption, the performance of adaptive execution model remained more or less similar to dynamic model. The average battery power consumption remained 15 mW (using D1) and 31 mW (using D2) as compared with dynamic execution model having 18 mW (using D1) and 36 mW (using D2) (see Figure 5.44). However, the energy spikes which were witnessed during dynamic execution were not found in adaptive execution. This phenomenon happened because dynamic model was bounded to execute few components in



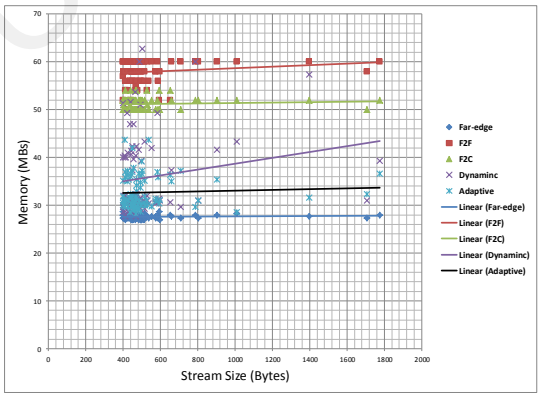
(a) Power Consumption Comparison of D1.



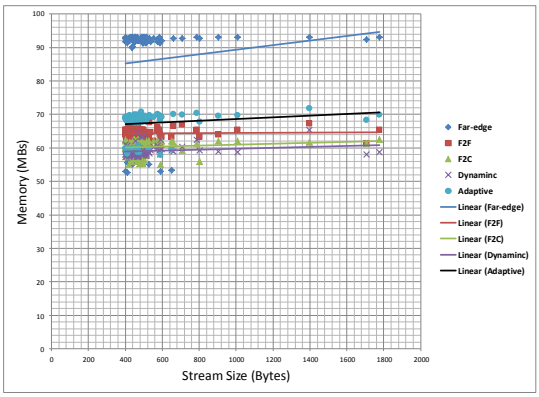
(b) Power Consumption Comparison of D2.

Figure 5.44: Power Consumption Comparisons - Static vs. Dynamic vs. Adaptive

far-edge devices regardless of the computational complexity. In contrast, the adaptive execution model enables multi-point data stream management and multi-point data stream offloading schemes in order to adaptively switch among execution modes from any point of execution in MDSM applications. Therefore, whenever the computational complexities increases and far-edge devices enter in critical situations, *i.e.* a few resources left, the execution model switches to other execution modes *i.e.* collaborative or remote. Hence the scheme conserves energy as well as adapts the seamless application execution behavior. Overall, the adaptive execution model consumes less battery power as compared to static execution in mobile edge servers and CC servers. In addition, the battery power consumption remains lower at most of the executions as compared to dynamic model.

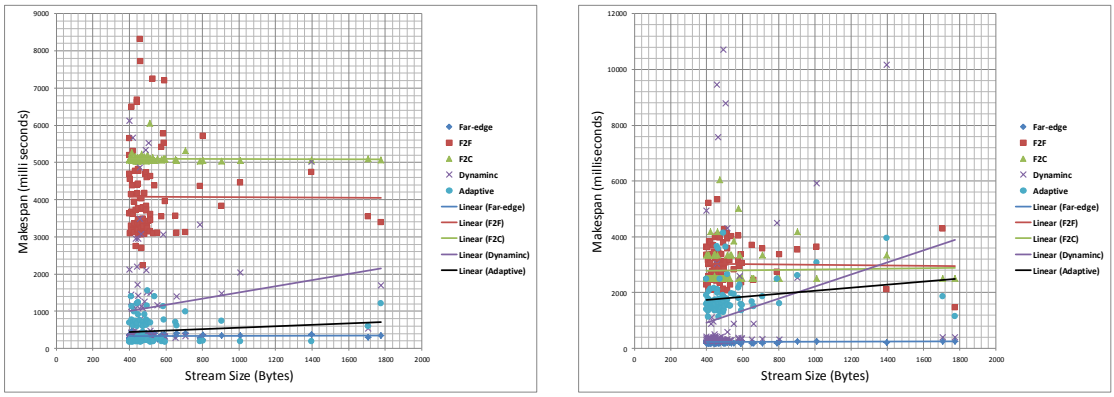


(a) Memory Consumption Comparison of D1.



(b) Memory Consumption Comparison of D2.

Figure 5.45: Memory Comparisons - Static vs. Dynamic vs. Adaptive



(a) Makespan Comparison of D1.

(b) Makespan Comparison of D2.

Figure 5.46: Makespan Comparisons - Static vs. Dynamic vs. Adaptive

Considering the memory consumption, the adaptive execution model performs comparatively equal or better as compared to static applications in far-edge devices and dynamic execution model. The memory consumption remained lower because most the executions either performed inside mobile devices or the model adaptively switched to other execution modes. Therefore, the average memory consumption of adaptive applications remained 33 MB and 67 MB as compared to dynamic applications which consumed 36 MB and 59 MB.

Although the adaptive execution model outperforms other models in terms of memory utilization and power consumption. The makespan trends vary significantly (see 5.46). For example, the makespan during adaptive application execution in D1 remained 468 ms on average which is far better than dynamic model having average makespan of 1119 ms. However, when same applications were executed in D2, the average makespan of adaptive applications (*i.e.* 1808 ms) far exceeds the average makespan of dynamic applications (*i.e.* 1196 ms). Therefore, it should be noted that makespan in adaptive applications remains flexible depending upon the availability of onboard computational resources, mobile edge servers and Internet connections for F2C communication.

5.4.1 Lessons Learned

Following lessons are learned after thorough performance evaluation, analysis and comparison of static, dynamic, and adaptive execution models.

- There is no high level of correlation among energy consumption, memory utilization, and execution when static applications were executed in far-edge devices (as witnessed in Section 5.3.3).
- During static application execution using F2F communication model, there was a medium level of correlation between memory utilization and makespan which shows that executions with high makespan may quickly hamper memory resources in far-edge devices. Therefore, MDSM applications must transfer the data stream to mobile edge servers or CC servers before overloading the memory resources.
- The static execution using F2C communication model shows that although a small level of correlation was observed between makespan and memory utilization, the data stream buffering in far-edge devices increases battery power consumption. In this case, the devices consume power for data buffering in mobile devices as well as the energy cost of data communication increases because of transferring large size of data streams. Therefore, MDSM applications were needed to control the data size as well as data rate in order to lower the memory utilization and cost of energy.
- The dynamic execution model gracefully addresses the issues of memory utilization and power consumption using transient data stream management and opportunistic data stream offloading schemes, In addition, the model enables maximum data processing using onboard resources in far-edge devices. This strategy enabled in lowering the makespan as well as reducing the dependency over Internet connection. In addition, the model reduced bandwidth utilization by processing up to 91%

data streams using far-edge devices and up to 98% data stream using far-edge devices and F2F communication model. Therefore, for about 2% to 6% data stream, the model depends upon Internet connections. Despite significant achievements, dynamic model is bounded for user intervention in order to set the file size thresholds which require laborious efforts of application developers to profile the file size for each far-edge device.

- The adaptive execution model addresses the issue of file sizes. In addition, it performs transient data stream management and opportunistic offloading operations only when the device enters in critical situations or unable to perform local executions. The adaptive model enables resource profiling and context monitoring at multiple points. Therefore, the application adaptively switches among different modes of execution. Depending upon the resource availability and computational requirements, each MDSM application adapts the execution behavior accordingly. The execution model still need improvement in terms of resource estimation, context profiling, and device profiling.

- Since the classifiers were trained on event-based data streams. Therefore, size of input data streams do not impacts the overall accuracy of classifiers as it is in the case of other data mining algorithms which need whole data streams inside memory in order to better predict (*i.e.* classifying, clustering, or finding association rules) the knowledge patterns. The accuracy of classifiers depends upon the personalization features of MDSM applications which need more research, therefore, left for future research work.

5.4.2 Qualitative Comparison of UniMiner

Due to the absence of a simulator to deploy similar systems for performance comparison and analysis, the proposed architecture was qualitatively compared with relevant systems as presented in Table 5.14. The comparison was made based on analytic mode, application execution model, data stream offloading strategies, and transient data management approach for system implementation.

Table 5.14: Qualitative Comparison of UniMiner.

Study	Analytic Mode	Execution	Data offloading	Data Management
Star	Mobile	Local	NA	No
CARDAP	Mobile/Cloud	Distributed	Push/Pull based	No
CC-Stream	Mobile	Distributed	Push based	No
Here-n-Now	Mobile/Cloud	Distributed	Push/pull based	No
PDM	Mobile	Distributed	Agent-based	No
MARS	Mobile	Local	NA	No
SOA	Server	Distributed	Push based	No
OMM	Mobile	Local	NA	No
ShareLikeCrowd	Mobile/Cloud	Distributed	push/pull based	No
UniMiner	Mobile/P2P/Cloud	Distributed	Dynamic/Adaptive	Yes

The UniMiner execution model operates in three tiers, whereas other systems either work independently or in two-layer settings. For example, OMM (P. D. Haghighi et al., 2013), CAROMM (Zaslavsky, Jayaraman, & Krishnaswamy, 2013), Star (Abdallah et al., 2015), and MARS (J. B. Gomes, Krishnaswamy, Gaber, Sousa, & Menasalvas, 2012b) are systems that work independently. Therefore, these systems adapt the execution behavior at the analytics component level. Comparatively, UniMiner enhances the system performance in mobile environments by enabling maximum execution without algorithm-level adaptations. On the other hand, cloud-based systems like CARDAP (Jayaraman, Gomes, et al., 2014) function at the mobile and cloud levels, or edge services are enabled through agent-oriented architectures like PDM (Gaber, Stahl, & Gomes, 2014) or cloud-based collaborative settings like MOSDEN (Jayaraman, Perera, et al., 2014). Nonetheless, the three-layer architecture of UniMiner supports generality by customizing the AllJoyn framework at the collaborative layer, which is expandable to thousands of

IoT systems (Framework, 2015).

In addition, existing systems transfer data streams as either push-based, pull-based or agent-based collaborative strategies. For example, the CC-Stream (J. B. B. Gomes, Gaber, Sousa, & Menasalvas, 2011) and SOA (Talia & Trunfio, 2010) operate as push-based systems, where the data streams are uploaded to a cloud environment and the cloud resource manager directs further executions. Other systems, such as MOSDEN (Jayaraman, Perera, et al., 2014) and CARDAP (Jayaraman, Gomes, et al., 2014) transfer data streams using push/pull-based strategies to work collaboratively in mobile cloud computing environments. Similarly, PDM (Gaber, Stahl, & Gomes, 2014) performs agent-based collaborative executions. Still, none of these systems execute dynamic data stream offloading to consider ground truth information for efficient data processing in MECC environments. Aside from this, the data stream offloading scheme in UniMiner ensures adaptive and collaborative execution of analytics components in the MECC architecture. In addition, the data stream offloading scheme effectively reduces memory and energy consumption when the results are compared with the simple push-based data transfer in both collaborative and cloud-based analytics. Furthermore, none of the existing systems handle online data streams to address the problems of information loss and complete data processing. However, UniMiner seamlessly manages the data stream in transient data stores, which increases efficiency in terms of makespan and enhances system performance by enabling heavy-weight analytic components.

5.5 Summary

This chapter presented the performance evaluation of static execution models and maps the existing systems accordingly. In addition, the chapter presented the performance evaluation results of proposed dynamic and adaptive execution models. The performance evaluation was made by developing a real-world use-case applications for activity detec-

tion in MECC environments. The models were evaluated in terms of memory utilization, battery power consumption, makespan, and accuracy of results produced by MDSM applications. The data reduction analysis of proposed models were also presented. The chapter presented the detailed correlation analysis, and comparisons of static, dynamic, and adaptive execution models. Finally, the chapter presented qualitative comparison of proposed architecture with existing systems.

CHAPTER 6: CONCLUSION AND FUTURE RESEARCH DIRECTIONS

This chapter concludes the thesis by discussing objectives of research and mapping the achievements accordingly. In addition, the chapter presents the future research issues in this thesis and presents the future research agenda accordingly.

6.1 Achievements

Primarily this research work had four objectives.

1. **OB1:** To conduct the experimental studies for feasibility and analysis of MDSM applications in far-edge mobile devices.

2. **OB2:** To design, verify, validate, and develop a three-tier MECC architecture for distributed MDSM applications.

3. **OB3:** To design and develop a dynamic execution model for distributed MDSM applications in MECC systems.

4. **OB4:** To design and develop an adaptive execution model for distributed MDSM applications in MECC systems.

The research objectives were achieved as follows.

In order to achieve OB1, a thorough performance evaluations of nine data stream mining algorithms were made using three far-edge mobile devices. The results were evaluated in terms of energy consumption, memory utilization, and makespan and presented in Chapter 3.

In order to achieve OB2, a three-tier component based architecture was proposed for MDSM application execution in MECC systems. The detailed discussion on layers and components was presented in Chapter 3. The proposed architecture was modeled and verified using HLPN, SMT-Lib, and Z3 solver. Moreover, the architecture was simulated

using PiPE+ editor and it was found that, if MDSM applications run statically without any controlling mechanism, the architecture faces the state explosion problem.

In order to achieve OB3 and OB4, chapter 4 presents the discussion on operations of dynamic and adaptive execution models. The models enables to handle the state explosion problem using data stream management and offloading schemes. The performance evaluation and comparison of proposed execution models with existing systems having static execution models are presented in Chapter 5.

6.2 Future Research Agenda

This thesis presents a novel idea of distributed mobile data stream mining in MECC systems whereby all application execution processes are device-centric. However, we perceive numerous research directions in order to further the research work from this thesis.

6.2.1 Future Research Work

The work presented in this thesis is expendable in following directions.

Multi-tier Architectures

The computing technologies are growing rapidly and next-generation MDSM platforms needs to use these processing technologies in order to accelerate the application performance. Despite of wide acceptance existing literature still lacks the multi-tier and heterogeneous data processing platforms. Therefore, future MDSM platforms should be designed with scalable topological settings using heterogeneous computing architectures blended with CPUs, GPUs, FPGAs, and large scale data centers. In addition, hierarchical memory architectures based on Caches, RAMs, and internal and external storage should also considered to design next-generation applications and platforms.

Load-balancing

Considering the advancements in computing technologies, future MDSM platforms will span across resource-constrained IoTs, wearable, and mobile devices at one end and

resourceful servers, clusters, and multi-cloud infrastructures on the other end. Future MDSM platforms need to integrate efficient load-balancing strategies in order to minimize the latency, efficient energy utilization, reduce bandwidth consumption and in-network data movement across the platforms. The new load-balancing strategies may integrate fuzzy logic and soft set theory based methods for improved efficiency. In addition, deep context models could be used in order to improve the load-balancing strategies across the platforms.

Optimization

The streaming data in mobile environments challenges the capacities of MDSM platforms in terms of energy consumption, storage management, bandwidth utilization, performance gain, privacy preservation, scheduling, and workflow management. Considering the above mentioned challenges, the MDSM applications and platforms need to be optimized for data processing, task scheduling, privacy preservation, and knowledge management. In addition with this the optimization algorithms should ensure seamless application execution across multiple devices and computing systems. The MDSM platforms should enable dynamic and adaptive application execution in MECC systems. TO further the research, the optimization strategies should be devised to achieve the maximum trade-off between data processing efforts and application execution in multiple platforms. Considering the optimization objectives, new algorithms must ensure the reduced and optimal resource consumption both for application execution and the resource required to execute the optimization algorithms itself.

Data Stream and Knowledge Management

MDSM applications need to handle the data streams in multiple formats and need different data management strategies. The MDSM platforms must provide the optimal data management schemes for raw data streams. To this end, existing in-memory data management schemes needs to be improved in order to efficiently handle the stream-

ing data considering its velocity, variety, volume, and variability characteristics. MDSM applications convert raw data streams into different formats at each stage of execution. These formats include raw data converted into event data streams, feature vectors, structured formats such as tables, to name a few. In addition, the intermediate data generated during data processing, when the data populated in data structures (i.e. arrays, trees, and graphs), challenge the computational capacities of resource constrained devices and computing systems which have low amount of available memory. New data management strategies are required to efficiently handle the intermediate data streams. Finally, the MDSM applications produce knowledge patterns which need to be integrated and summarized for a holistic view of incoming data streams. Future MDSM platforms must provide synchronized knowledge management schemes across the MECC systems.

Programming Models, Design Patterns, and Development Environments

Considering the heterogeneity in next-generation MDSM applications and platforms, new programming models, design patterns, and development environments are needed. Existing simulation tools and programming models support application execution as either mobile-first or cloud-first approach, however, new programming models should support the application execution across MECC systems. In addition, new design patterns are required which could be reused to each the application development process in MECC systems. Moreover, new integrated development environments (IDEs) are needed to integrate the programming models and design patterns. The IDEs should provide support for drop and drop component based visual workflow management across MECC systems. Further, the IDEs should provide reusable components for rapid application development in MECC systems.

6.2.2 Future Research Areas

This section presents some future areas in order to accelerate the research work in mobile data stream mining applications and platforms. Due to application and platform level heterogeneity, MDSM applications can help in future and emerging research areas in multiple ways. Figure 6.1 depicts the relevance of this thesis with future research areas.

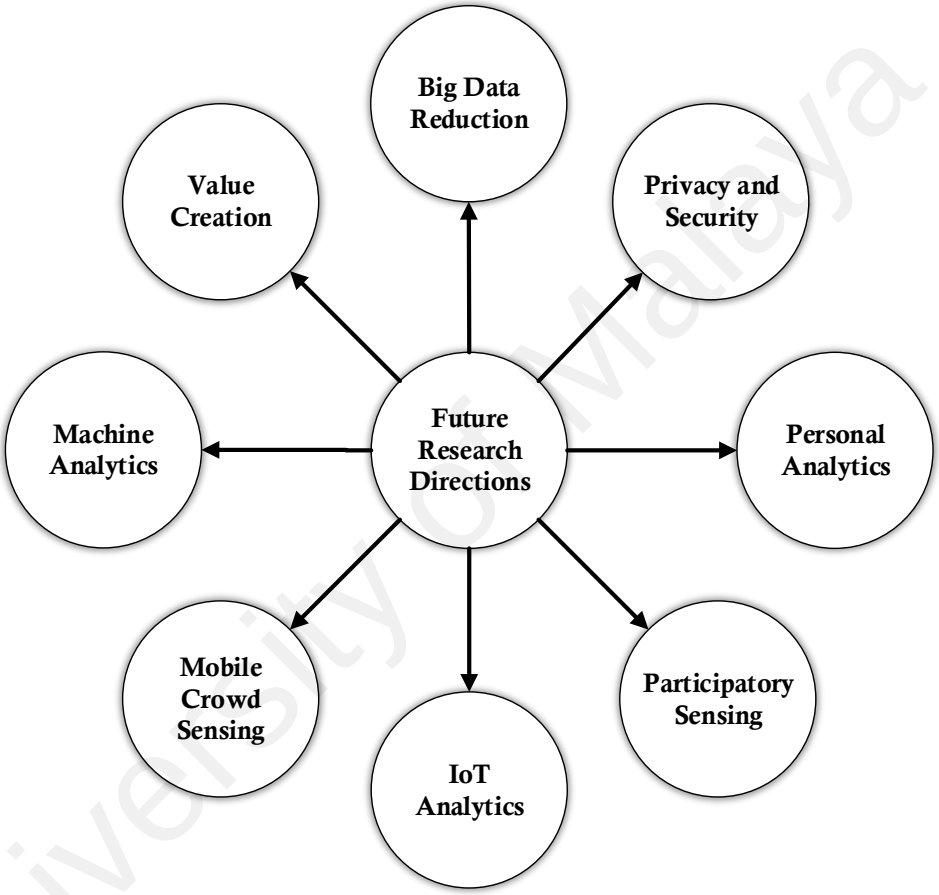


Figure 6.1: Future Research Directions.

Privacy and Security: The onboard data sources in far-edge devices produce personal data streams, therefore, mobile data stream mining applications need to address the privacy and security concerns of end users (Sokolova & Matwin, 2016). However, to this end, existing literature lacks in scalable end-to-end privacy preservation models for MDSM applications in MECC systems (Jayaraman, Gomes, et al., 2014). The privacy preservation models are needed to be designed and embedded in existing MDSM

applications without loosing the quality of uncovered knowledge patterns. Moreover, the MDSM applications should enable secure data and knowledge transfer strategies for data movement inside MDSM platforms. To this end, privacy and security challenges need serious attentions in order to prevail this important research area.

Big Data Reduction: The continuous evolution in mobile data streams eventually results in big data. However, analyzing the massive amount of data and uncovering useful patterns for end users is a challenging task. The deployment of MDSM applications at user-end can help in reducing big data wherein the users can uncover the knowledge patterns using personal far-edge devices (Rehman & Batool, 2015; Rehman, Chang, Batool, & Wah, 2016). Existing literature lacks the pattern based data sharing strategies for big data systems. Future research work should focus on the development and deployment of learning models complying with the needs of big data systems. In addition, pattern sharing, knowledge summarization, and big data aggregation models are needed in order to deal with reduced big data. In essence pattern based big data reduction can benefit to users and big data system providers in many ways including, *a*) reduced data communication cost, *b*) minimum bandwidth utilization, *c*) reduced in-network data movement, *d*) fewer efforts in data cleaning and preprocessing for conversion of unstructured big data in to structured datasets, and last but not the least, *e*) big data system providers can offer personalized services to end users.

Value Creation The MDSM applications in MECC systems can help in value creation for customers and enterprises in multiple ways. At one end, the customers can use the personal far-edge devices, edge servers, and cloud computing systems to find the personal knowledge patterns. At the other end, enterprises can acquire the customers' data in order to develop and optimize their business process models and meet their needs (Rehman, Chang, et al., 2016). MDSM applications can benefit in value creation for a wide spectrum of user-centered business models.

Machine Analytics: The MDSM applications can benefit in machine analytics in order to uncover the operating and performance behaviors of machines. The embedded data stream mining components in machines can help in onboard and off-board data collection and uncovering machine behaviors in MECC systems. For example, in manufacturing industries, large scale industrial production units can use embedded data stream mining components to uncover knowledge patterns from machine log files and monitor the machine's performance (Cochran, Kinard, & Bi, 2016). Similarly, local and collective intelligence in robotics can be embedded using MDSM applications. Few more example application areas include smart cars, vehicular ad-hoc networks, machine to machine communication systems, and cyber-physcial systems.

Personal Analytics: Mobile users generate personal data from a plethora of sensory and non-sensory data sources (Rehman et al., 2015). These data sources collect data streams of mobile users from onboard and off-board sensors as well as the data generated in the result of user interactions with mobile devices, physical activities performed by users, and the behavioral data of users on social networks and world wide web. The MDSM applications in MECC system can help in uncovering personal knowledge patterns from above mentioned personal data. The knowledge patterns are useful for lifestyle and wellness management applications, behavioral analytic driven systems, mobile health applications, mobile social networks, and mobile commerce, to name a few.

IoT Analytics: The MDSM applications can be embedded in IoT systems in order to uncover the device-centric and collective knowledge patterns. The applications can be deployed in a single device and multi-device settings. In single device settings, the uncovered knowledge patterns could be used for improving single device usage experiences however in the case of multi-device settings, the patterns could be used for the overall improvement of IoT systems (Satyanarayanan et al., 2015). In addition, the application logic could be distributed across multiple IoT devices in order to find the collective behavior.

Mobile Crowd Sensing: The MDSM applications in MECC systems can facilitate in mobile crowd sensing systems (Jayaraman, Gomes, et al., 2014). For example, the data streams collected by smart city management applications for traffic management, commuters facilitation, crowd management in sporting arenas, and facilitating pilgrims and peoples gatherings at holy places. Similarly, MDSM applications can facilitate in managing crowds of animals, vehicles, IoTs and many more similar applications.

Participatory Sensing: Participatory sensing is another application area for MDSM applications and platform (Jayaraman, Perera, et al., 2014). The knowledge patterns generated by mobile users can help governments, businesses, enterprises, corporations, and third party public data stream collectors in order to develop user-driven applications and systems. However, participatory sensing systems must ensure user privacy and security of shared data. In addition, new incentive mechanisms are needed in order to lure mobile users for participatory data sharing.

In this section, we discussed a few future research opportunities for the intervention of MDSM applications and platforms. However, the tremendous growth in IoTs, big data, cloud computing, and mobile edge computing has risen many new application areas and research opportunities for MDSM applications and platforms. Therefore, we perceive that the deployment of MDSM applications in MECC system will quickly prevail in all sectors of the economy and humane lifestyle management.

6.3 Final Thoughts

This research mainly focuses MDSM applications for personal data. Therefore the proposed three-tier architecture in this thesis enables device-centric approach that benefits in delegating complete control of data processing at user and device ends. In addition, the adoption of far-edge devices as primary platform for data processing not only reduces makespan and size of data streams in MDSM applications. It also enables maximum data

processing at user end which lowers the risks of compromises on privacy and security of personal data. In order to scale up the architecture further research is required and we are aiming to extend this work in multiple dimensions as discussed in Section 6.2.

The dynamic execution model in this thesis facilitates in opportunistic execution of MDSM applications in three-tier architecture. The model meets the objective of maximum data processing near the data sources however it currently supports the data-intensive applications having low computational complexities. In addition, the execution model supports controlled size of input data stream. The model could be further improved by setting dynamic upper size of data stream which could be derived from correlation analysis between input data streams and quality of knowledge patterns. In addition, resource estimation scheme of dynamic execution model could be further improved using statistical estimation methods.

The adaptive execution model supports data-intensive and compute-intensive applications. The strength of proposed model is its ability to perform data stream management and data stream offloading from multiple points of execution in MDSM applications. Another important feature is the ability of MDSM applications to adapt the execution behaviors according to underlying resources. This adaptation benefits in overcoming the issue of abrupt resource utilization (*i.e.* sudden energy and memory spikes). Using adaptive execution model, MDSM applications switch to collaborative or remote data processing modes at early stages of execution if onboard resources are not available.

To limit the scope, this thesis primarily presented the performance evaluation of proposed architecture and executions models with classification applications. We will extend this work in future by deploying association rule mining and clustering applications in proposed models. This thesis is just the beginning of next generation MDSM applications for personal data and big data reduction but we aim to extend this work to many interesting applications areas as presented in Section 6.2.

LIST OF PUBLICATIONS

1. Rehman, M. H., Liew, C. S., & Wah, T. Y. (2014). Frequent pattern mining in mobile devices: A feasibility study. In Information technology and multimedia (ICIMu), 2014 international conference on (pp. 351–356).
2. Rehman, M. H., Liew, C. S., & Wah, T. Y. (2014). UniMiner: Towards a unified framework for data mining. In Information and Communication Technologies (WICT), 2014 Fourth World Congress on (pp. 134–139).
3. Rehman, M. H., Liew, C. S., Wah, T. Y., Shuja, J., Daghighi, B., et al. (2015). Mining personal data using smartphones and wearable devices: A survey. Sensors, 15(2), 4430–4469.
4. Rehman, M. H., Chang, V., Batool, A., & Wah, T. Y. (2016), Big data reduction framework for value creation in sustainable enterprises, International Journal of Information Management, Volume 36, Issue 6, Part A, December 2016, 917-928
5. Rehman, M. H., Liew, C. S., , Iqbal, A., Wah, T. Y., & Jayaraman, P. P. (13-17 June, 2016). Opportunistic computation offloading in mobile edge cloud computing environments. in 17th IEEE international conference on mobile data management (pp. 208-213).
6. Rehman, M. H., Liew, C. S., Wah, T. Y., & Khan, M. K. (2017). Towards Next-generation Mobile Data Stream Mining Applications: Opportunities, Challenges, and Future Research Directions. Journal of Network and Computer Applications, 79, pp. 1-24.
7. Rehman, M. H., Liew, C. S., Batool, A., Wah, T. Y., & Khan, A.R. (2017). Execution Models for MDA Applications in MECC Systems. IEEE IT Professional.

REFERENCES

Abdallah, Z. S., Gaber, M. M., Srinivasan, B., & Krishnaswamy, S. (2012). Cbars: Cluster based classification for activity recognition systems. In *Advanced machine learning technologies and applications* (pp. 82–91). Springer.

Abdallah, Z. S., Gaber, M. M., Srinivasan, B., & Krishnaswamy, S. (2015). Adaptive mobile activity recognition system with evolving data streams. *Neurocomputing*, *150*, 304–317.

Abolfazli, S., Sanaei, Z., Ahmed, E., Gani, A., & Buyya, R. (2014). Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges. *Communications Surveys & Tutorials, IEEE*, *16*(1), 337–368.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, vldb* (Vol. 1215, pp. 487–499).

Ahmad, A., & Ahmad, E. (2016). A survey on mobile edge computing. In *10th internatonal conference on intelligenct systems and control (isco), coimbatore, india.*

Ahmad, R. W., Gani, A., Hamid, S. H. A., Shiraz, M., Yousafzai, A., & Xia, F. (2015). A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, *52*, 11–25.

Ahmed, E., Akhunzada, A., Whaiduzzaman, M., Gani, A., Ab Hamid, S. H., & Buyya, R. (2015). Network-centric performance analysis of runtime application migration in mobile cloud computing. *Simulation Modelling Practice and Theory*, *50*, 42–56.

Al-Fuqaha, A., Khreishah, A., Guizani, M., Rayes, A., & Mohammadi, M. (2015). Toward better horizontal integration among iot services. *IEEE Communications Magazine*, *53*(9), 72–79.

Altomare, A., Cesario, E., Comito, C., Marozzo, F., & Talia, D. (2013). Using clouds for smart city applications. In *Cloud computing technology and science (cloudcom), 2013 ieee 5th international conference on* (Vol. 2, pp. 234–237).

Amini, A., Wah, T. Y., & Saboohi, H. (2014). On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology*, *29*(1), 116–141.

Android (operating system). (2016, 03). Retrieved from `https://en.wikipedia.org/wiki/Android_(operating_system)`

Apple iphone history. (2016, 03). Retrieved from `apple-history.com/iPhone`

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... Stoica, I. (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58.

Arunkumar, S., Srivatsa, M., & Rajarajan, M. (2015). A review paper on preserving privacy in mobile environments. *Journal of Network and Computer Applications*,

53, 74–90.

Augustyniak, P., Smoleń, M., Mikrut, Z., & Kańtoch, E. (2014). Seamless tracing of human behavior using complementary wearable and house-embedded sensors. *Sensors*, *14*(5), 7831–7856.

Bache, K., & Lichman, M. (2013). Uci machine learning repository. *URL http://archive. ics. uci. edu/ml*, *19*.

Bahl, V. (2015, May). *The emergence of micro datacenters (cloudlets) for mobile computing*. Retrieved from `http://research.microsoft.com/apps/video/default.aspx?id=246447`

Bittencourt, L. F., Lopes, M. M., Petri, I., & Rana, O. F. (2015). Towards virtual machine migration in fog computing. In *2015 10th international conference on p2p, parallel, grid, cloud and internet computing (3pgcic)* (pp. 1–8).

Bolboaca, S.-D., & Jäntschi, L. (2006). Pearson versus spearman, kendall tau correlation analysis on structure-activity relationships of biologic active compounds. *Leonardo Journal of Sciences*, *5*(9), 179–200.

Bonet, P., Lladó, C. M., Puijaner, R., & Knottenbelt, W. J. (2007). Pipe v2. 5: A petri net tool for performance modelling. In *Proc. 23rd latin american conference on informatics (clei 2007).*

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the mcc workshop on mobile cloud computing* (pp. 13–16).

Boukhechba, M., Bouzouane, A., Bouchard, B., Gouin-Vallerand, C., & Giroux, S. (2015). Online prediction of peoples next point-of-interest, concept drift support. In *Human behavior understanding* (pp. 97–116). Springer.

Braojos, R., Beretta, I., Constantin, J., Burg, A., & Atienza, D. (2014). A wireless body sensor network for activity monitoring with low transmission overhead. In *Embedded and ubiquitous computing (euc), 2014 12th ieee international conference on* (pp. 265–272).

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, *25*(6), 599–616.

Cameron, J. J., Cuzzocrea, A., Jiang, F., & Leung, C. K. (2013). Mining frequent itemsets from sparse data streams in limited memory environments. In *Web-age information management* (pp. 51–57). Springer.

Chen, J., & Chen, P. (2014). Sequential pattern mining for uncertain data streams using sequential sketch. *Journal of Networks*, *9*(2), 252–258.

Choe, E. K., Lee, N. B., Lee, B., Pratt, W., & Kientz, J. A. (2014). Understanding quantified-selfers' practices in collecting and exploring personal data. In *Proceedings of the 32nd annual acm conference on human factors in computing systems* (pp. 1143–1152).

Ciurana, E. (2009). *Developing with google app engine*. Apress.

Cochran, D. S., Kinard, D., & Bi, Z. (2016). Manufacturing system design meets big data analytics for continuous improvement. *Procedia CIRP*, *50*, 647–652.

Comito, C., & Talia, D. (2013). Energy characterization of data mining algorithms on mobile devices. In *Energy efficiency in large scale distributed systems* (pp. 98–113). Springer.

Cord, M., & Cunningham, P. (2008). *Machine learning techniques for multimedia: case studies on organization and retrieval*. Springer Science & Business Media.

Cormode, G., Garofalakis, M., Haas, P. J., & Jermaine, C. (2012). Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, *4*(1–3), 1–294.

Cormode, G., & Muthukrishnan, S. (2004). An improved data stream summary: The count-min sketch and its applications. In *Latin 2004: Theoretical informatics* (pp. 29–38). Springer.

De Moura, L., & Bjørner, N. (2008). Z3: An efficient smt solver. In *International conference on tools and algorithms for the construction and analysis of systems* (pp. 337–340).

De Moura, L., & Bjørner, N. (2009). Satisfiability modulo theories: An appetizer. In *Formal methods: Foundations and applications* (pp. 23–36). Springer.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1–38.

Developers, A. (2014). Get the android sdk. Retrieved from `https://developer.android.com/sdk/index.html`

Diaz, M. (2013). *Petri nets: fundamental models, verification and applications*. John Wiley & Sons.

Dogan, N., & Tanrikulu, Z. (2013). A comparative analysis of classification algorithms in data mining for accuracy, speed and robustness. *Information Technology and Management*, *14*(2), 105–124.

Donohoo, B. K., Ohlsen, C., Pasricha, S., Xiang, Y., & Anderson, C. (2014). Context-aware energy enhancements for smart mobile devices. *Mobile Computing, IEEE Transactions on*, *13*(8), 1720–1732.

Dou, A. J., Kalogeraki, V., Gunopulos, D., Mielikinen, T., Tuulos, V., Foley, S., & Yu, C.

(2011). Data clustering on a network of mobile smartphones. In *Applications and the internet (saint), 2011 ieee/ipsj 11th international symposium on* (pp. 118–127).

Drolia, U., Martins, R. P., Tan, J., Chheda, A., Sanghavi, M., Gandhi, R., & Narasimhan, P. (2013). The case for mobile edge-clouds. In *Ubiquitous intelligence and computing, 2013 ieee 10th international conference on and 10th international conference on autonomic and trusted computing (uic/atc)* (pp. 209–215).

Eom, H., Figueiredo, R., Cai, H., Zhang, Y., & Huang, G. (2015). Malmos: Machine learning-based mobile offloading scheduler with online training. In *Mobile cloud computing, services, and engineering (mobilecloud), 2015 3rd ieee international conference on* (pp. 51–60).

Fernando, N., Loke, S. W., & Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*, *29*(1), 84–106.

Ferreira, H., Duarte, S., & Preguiça, N. (2010). 4sensing–decentralized processing for participatory sensing data. In *Parallel and distributed systems (icpads), 2010 ieee 16th international conference on* (pp. 306–313).

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine learning*, *2*(2), 139–172.

Framework, A. (2015). *Allseen alliance.* Retrieved from `https://allseenalliance.org/framework/documentation/learn`

Gaber, M. M. (2009). Data stream mining using granularity-based approach. In *Foundations of computational, intelligencevolume 6* (pp. 47–66). Springer.

Gaber, M. M., Gama, J., Krishnaswamy, S., Gomes, J. B., & Stahl, F. (2014). Data stream mining in ubiquitous environments: state-of-the-art and current directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *4*(2), 116–138.

Gaber, M. M., Stahl, F., & Gomes, J. B. (2014). Pocket data mining framework. In *Pocket data mining* (pp. 23–40). Springer.

Gama, J. (2013). Data stream mining: the bounded rationality. *Informatica*, *37*(1).

Goldberg, A. B., Zhu, X., Singh, A., Xu, Z., & Nowak, R. (2009). Multi-manifold semi-supervised learning. *in Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS) 2009, Clearwater Beach, Florida, USA.*.

Gomes, J. B., Krishnaswamy, S., Gaber, M. M., Sousa, P. A., & Menasalvas, E. (2012a). Mars: a personalised mobile activity recognition system. In *Mobile data management (mdm), 2012 ieee 13th international conference on, july 23-26, balngluru, india* (pp. 316–319).

Gomes, J. B., Krishnaswamy, S., Gaber, M. M., Sousa, P. A., & Menasalvas, E. (2012b). *Mobile activity recognition using ubiquitous data stream mining*. Springer.

Gomes, J. B. B., Gaber, M. M., Sousa, P. A., & Menasalvas, E. (2011). Context-aware collaborative data stream mining in ubiquitous devices. In *Advances in intelligent data analysis x* (pp. 22–33). Springer.

Gordon, M., Zhang, L., Tiwana, B., Dick, R., Mao, Z., & Yang, L. (2013). *Powertutor: a power monitor for android-based mobile platforms.*

Gu, T., Wang, L., Wu, Z., Tao, X., & Lu, J. (2011). A pattern mining approach to sensor-based human activity recognition. *Knowledge and Data Engineering, IEEE Transactions on*, *23*(9), 1359–1372.

Guan, T., Wang, Y., Duan, L., & Ji, R. (2015). On-device mobile landmark recognition using binarized descriptor with multifeature fusion. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *7*(1), 12.

Ha, K., Chen, Z., Hu, W., Richter, W., Pillai, P., & Satyanarayanan, M. (2014). Towards wearable cognitive assistance. In *Proceedings of the 12th annual international conference on mobile systems, applications, and services* (pp. 68–81).

Ha, K., & Satyanarayanan, M. (2015). Openstack++ for cloudlet deployment. *School of Computer Science Carnegie Mellon University Pittsburgh*.

Haghighi, P., Gaber, M., Krishnaswamy, S., & Zaslavsky, A. (2007). An architecture for context-aware adaptive data stream mining. In *Proceedings of the international workshop on knowledge discovery from ubiquitous data streams (iwkduds07).*

Haghighi, P. D., Krishnaswamy, S., Zaslavsky, A., Gaber, M. M., Sinha, A., & Gillick, B. (2013). Open mobile miner: A toolkit for building situation-aware data mining applications. *Journal of Organizational Computing and Electronic Commerce*, *23*(3), 224–248.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, *11*(1), 10–18.

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, *8*(1), 53–87.

Hartigan, J. A., & Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, *28*(1), 100–108.

Hassan, M. A., Wei, Q., & Chen, S. (2015). Elicit: Efficiently identify computation-intensive tasks in mobile applications for offloading. In *Networking, architecture and storage (nas), 2015 ieee international conference on* (pp. 12–22).

Hromic, H., Le Phuoc, D., Serrano, M., Antonic, A., Zarko, I. P., Hayes, C., & Decker, S. (2015). Real time analysis of sensor data for the internet of things by means of clustering and event processing. In *Communications (icc), 2015 ieee international conference on* (pp. 685–691).

Huang, G., Song, S., Gupta, J. N., & Wu, C. (2014). Semi-supervised and unsupervised extreme learning machines. *Cybernetics, IEEE Transactions on*, *44*(12), 2405–2417.

Jayaraman, P. P., Gomes, J. B., Nguyen, H. L., Abdallah, Z. S., Krishnaswamy, S., & Zaslavsky, A. (2014). Cardap: A scalable energy-efficient context aware distributed mobile data analytics platform for the fog. In *Advances in databases and information systems* (pp. 192–206).

Jayaraman, P. P., Perera, C., Georgakopoulos, D., & Zaslavsky, A. (2014). Mosden: A scalable mobile collaborative platform for opportunistic sensing applications. *arXiv preprint arXiv:1405.5867*.

Jayaraman, P. P., Sinha, A., Sherchan, W., Krishnaswamy, S., Zaslavsky, A., Haghighi, P. D., ... Do, M. T. (2012). Here-n-now: A framework for context-aware mobile crowdsensing. In *Proc. of the tenth international conference on pervasive computing*.

Kargupta, H., Gilligan, M., Puttagunta, V., Sarkar, K., Klein, M., Lenzi, N., & Johnson, D. (2010). Minefleet: The vehicle data stream mining system for ubiquitous environments. In *Ubiquitous knowledge discovery* (pp. 235–254). Springer.

Khan, A. M., Lee, Y.-K., Lee, S., & Kim, T.-S. (2010). Human activity recognition via an accelerometer-enabled-smartphone using kernel discriminant analysis. In *Future information technology (futuretech), 2010 5th international conference on* (pp. 1–6).

Khan, A. M., Siddiqi, M. H., & Lee, S.-W. (2013). Exploratory data analysis of acceleration signals to select light-weight and accurate features for real-time activity recognition on smartphones. *Sensors*, *13*(10), 13099–13122.

Khan, A. R., Othman, M., Madani, S. A., & Khan, S. U. (2014). A survey of mobile cloud computing application models. *IEEE Communications Surveys & Tutorials*, *16*(1), 393–413.

Khan, M. A. (2015). A survey of computation offloading strategies for performance improvement of applications running on mobile devices. *Journal of Network and Computer Applications*, *56*, 28–40.

Klas, G. I. (2015). Fog computing and mobile edge cloud gain momentum open fog consortium, etsi mec and cloudlets. Retrieved from `http://yucianga.info/?p=938`

Krishnaswamy, S., Gaber, M., Harbach, M., Hugues, C., Sinha, A., Gillick, B., ... Zaslavsky, A. (2009). Open mobile miner: a toolkit for mobile data stream mining. *ACM KDD09*.

Krishnaswamy, S., Gama, J., & Gaber, M. M. (2012). Mobile data stream mining: from algorithms to applications. In *Mobile data management (mdm), 2012 ieee 13th international conference on* (pp. 360–363).

Kumar, K., Liu, J., Lu, Y.-H., & Bhargava, B. (2013). A survey of computation offloading

for mobile systems. *Mobile Networks and Applications*, *18*(1), 129–140.

Larose, D. T. (2014). *Discovering knowledge in data: an introduction to data mining*. John Wiley & Sons.

Li, J., Peng, Z., Xiao, B., & Hua, Y. (2015). Make smartphones last a day: Pre-processing based computer vision application offloading. In *Sensing, communication, and networking (secon), 2015 12th annual ieee international conference on* (pp. 462–470).

Liang, Y., Zhou, X., Yu, Z., & Guo, B. (2014). Energy-efficient motion related activity recognition on mobile devices for pervasive healthcare. *Mobile Networks and Applications*, *19*(3), 303–317.

Lin, C., Choy, K.-l., Pang, G., & Ng, M. T. (2013). A data mining and optimization-based real-time mobile intelligent routing system for city logistics. In *Industrial and information systems (iciis), 2013 8th ieee international conference on* (pp. 156–161).

Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., & Qureshi, A. (2015). Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, *48*, 99–117.

Liu, P., Chen, Y., Tang, W., & Yue, Q. (2012). Mobile weka as data mining tool on android. In *Advances in electrical engineering and automation* (pp. 75–80). Springer.

Lowd, D., & Domingos, P. (2005). Naive bayes models for probability estimation. In *Proceedings of the 22nd international conference on machine learning* (pp. 529–536).

Lu, H., Frauendorfer, D., Rabbi, M., Mast, M. S., Chittaranjan, G. T., Campbell, A. T., ... Choudhury, T. (2012). Stresssense: Detecting stress in unconstrained acoustic environments using smartphones. In *Proceedings of the 2012 acm conference on ubiquitous computing* (pp. 351–360).

Luan, T. H., Gao, L., Li, Z., Xiang, Y., & Sun, L. (2015). Fog computing: Focusing on mobile users at the edge. *arXiv preprint arXiv:1502.01815*.

Miller, G. (2012). The smartphone psychology manifesto. *Perspectives on Psychological Science*, *7*(3), 221–237.

Min, J.-K., & Cho, S.-B. (2011). Activity recognition based on wearable sensors using selection/fusion hybrid ensemble. In *Systems, man, and cybernetics (smc), 2011 ieee international conference on* (pp. 1319–1324).

Mukherji, A., Srinivasan, V., & Welbourne, E. (2014). Adding intelligence to your mobile device via on-device sequential pattern mining. In *Proceedings of the 2014 acm international joint conference on pervasive and ubiquitous computing: Adjunct publication* (pp. 1005–1014).

Norusis, M. (2008). *Spss 16.0 statistical procedures companion*. Prentice Hall Press.

Oneto, L., Ghio, A., Ridella, S., & Anguita, D. (2015). Learning resource-aware classifiers for mobile devices: From regularization to energy efficiency. *Neurocomputing*, *169*, 225–235.

Ortiz, J., Huang, C.-C., & Chakraborty, S. (2015). Get more with less: Near real-time image clustering on mobile phones. *arXiv preprint arXiv:1512.02972*.

Oshin, T. O., Poslad, S., & Zhang, Z. (2015). Energy-efficient real-time human mobility state classification using smartphones. *Computers, IEEE Transactions on*, *64*(6), 1680–1693.

Ozzie, R. E., Gates III, W. H., Flake, G. W., Bergstraesser, T. F., Blinn, A. N., Brumme, C. W., ... Glasgow, D. A. (2011, April). *Personal data mining.* Google Patents. (US Patent 7,930,197)

Pasricha, S., Donohoo, B. K., & Ohlsen, C. (2015). A middleware framework for application-aware and user-specific energy optimization in smart mobile devices. *Pervasive and Mobile Computing*, *20*, 47–63.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Rai, A., Yan, Z., Chakraborty, D., Wijaya, T. K., & Aberer, K. (2012). Mining complex activities in the wild via a single smartphone accelerometer. In *Proceedings of the sixth international workshop on knowledge discovery from sensor data* (pp. 43–51).

Rehman, M. H., & Batool, A. (2015). The concept of pattern based data sharing in big data environments. *International Journal of Database Theory and Application*, *8*(4), 11–18.

Rehman, M. H., Chang, V., Batool, A., & Wah, T. Y. (2016). Big data reduction framework for value creation in sustainable enterprises. *International Journal of Information Management*, *36*(6, Part A), 917 - 928. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0268401216303097` doi: http://dx.doi.org/10.1016/j.ijinfomgt.2016.05.013

Rehman, M. H., Liew, C. S., & Wah, T. Y. (2014a). Frequent pattern mining in mobile devices: A feasibility study. In *Information technology and multimedia (icimu), 2014 international conference on* (pp. 351–356).

Rehman, M. H., Liew, C. S., & Wah, T. Y. (2014b). Uniminer: Towards a unified framework for data mining. In *Information and communication technologies (wict), 2014 fourth world congress on* (pp. 134–139).

Rehman, M. H., Liew, C. S., Wah, T. Y., Iqbal, A., & Jayaraman, P. P. (2016). Opportunistic computation offloading in mobile edge cloud computing environments. In *2016 17th ieee international conference on mobile data management (mdm)* (Vol. 1, pp. 208–213).

Rehman, M. H., Liew, C. S., Wah, T. Y., Shuja, J., & Daghighi, B. (2015). Mining personal data using smartphones and wearable devices: A survey. *Sensors*, *15*(2), 4430–4469.

Rehman, M. H., Sun, L. C., Wah, T. Y., & Khan, M. K. (2017). Towards next-generation heterogeneous mobile data stream mining applications: opportunities, challenges, and future research directions. *Journal of Network and Computer Applications*, *79*, 1-24. doi: 10.1016/j.jnca.2016.11.031

Samsung. (2014, February). *Samsung unveils galaxy s5 and new gear range.* Online. Retrieved from `http://www.samsung.com/uk/discover/mobile/samsung-unveils-galaxy-s5-and-new-gear-range/`

Satyanarayanan, M. (2015). A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets. *ACM SIGMOBILE Mobile Computing and Communications Review*, *18*(4), 19–23.

Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, *8*(4), 14–23.

Satyanarayanan, M., Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., ... Amos, B. (2015). Edge analytics in the internet of things. *IEEE Pervasive Computing*(2), 24–31.

Shaukat, U., Ahmed, E., Anwar, Z., & Xia, F. (2015). Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges. *Journal of Network and Computer Applications*.

Sherchan, W., Jayaraman, P. P., Krishnaswamy, S., Zaslavsky, A., Loke, S., & Sinha, A. (2012). Using on-the-move mining for mobile crowdsensing. In *Mobile data management (mdm), 2012 ieee 13th international conference on* (pp. 115–124).

Shi, Y., Zhang, L., Tian, Y., & Li, X. (2015). Data mining and knowledge management. In *Intelligent knowledge* (pp. 1–11). Springer.

Shoaib, M., Bosch, S., Incel, O. D., Scholten, H., & Havinga, P. J. (2014). Fusion of smartphone motion sensors for physical activity recognition. *Sensors*, *14*(6), 10146–10176.

Sidek, K. A., Mai, V., & Khalil, I. (2014). Data mining in mobile ecg based biometric identification. *Journal of Network and Computer Applications*, *44*, 83–91.

Siirtola, P., & Röning, J. (2012). Recognizing human activities user-independently on smartphones based on accelerometer data. *International Journal of Interactive Multimedia and Artificial Intelligence*, *1*(5).

Siirtola, P., & Roning, J. (2013). Ready-to-use activity recognition for smartphones. In *Computational intelligence and data mining (cidm), 2013 ieee symposium on* (pp. 59–64).

Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., & Satyanarayanan, M. (2013). Scalable crowd-sourcing of video from mobile devices. In *Proceeding of the 11th annual international conference on mobile systems, applications, and services* (pp. 139–152).

Sokolova, M., & Matwin, S. (2016). Personal privacy protection in time of big data. In *Challenges in computational statistics and data mining* (pp. 365–380). Springer.

Srinivasan, V., Moghaddam, S., Mukherji, A., Rachuri, K. K., Xu, C., & Tapia, E. M. (2014). Mobileminer: Mining your frequent patterns on your phone. In *Proceedings of the 2014 acm international joint conference on pervasive and ubiquitous computing* (pp. 389–400).

Stahl, F., Gaber, M. M., Aldridge, P., May, D., Liu, H., Bramer, M., & Philip, S. Y. (2012). Homogeneous and heterogeneous distributed classification for pocket data mining. In *Transactions on large-scale data-and knowledge-centered systems v* (pp. 183–205). Springer.

Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P., & Pastrana, S. (2015). Power-aware anomaly detection in smartphones: An analysis of on-platform versus externalized operation. *Pervasive and Mobile Computing*, *18*, 137–151.

Swan, M. (2012a). Health 2050: The realization of personalized medicine through crowdsourcing, the quantified self, and the participatory biocitizen. *Journal of Personalized Medicine*, *2*(3), 93–118.

Swan, M. (2012b). Sensor mania! the internet of things, wearable computing, objective metrics, and the quantified self 2.0. *Journal of Sensor and Actuator Networks*, *1*(3), 217–253.

Talia, D., & Trunfio, P. (2010). Mobile data mining on small devices through web services. *Mobile Intelligence*, *69*, 264.

Triguero, I., García, S., & Herrera, F. (2015). Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, *42*(2), 245–284.

Wang, L., Gu, T., Tao, X., & Lu, J. (2012). A hierarchical approach to real-time activity recognition in body sensor networks. *Pervasive and Mobile Computing*, *8*(1), 115–130.

Wang, N., Merrett, G. V., Maunder, R. G., & Rogers, A. (2013). Energy and accuracy trade-offs in accelerometry-based activity recognition. In *Computer communications and networks (icccn), 2013 22nd international conference on* (pp. 1–6).

Wickramasinghe, A., & Ranasinghe, D. C. (2015). Recognising activities in real time using body worn passive sensors with sparse data streams: To interpolate or not to interpolate? In *proceedings of the 12th eai international conference on mobile and ubiquitous systems: Computing, networking and services on 12th eai international conference on mobile and ubiquitous systems: Computing, networking and services* (pp. 21–30).

Wu, P., Zhu, J., & Zhang, J. Y. (2013). Mobisens: A versatile mobile sensing platform for real-world applications. *Mobile Networks and Applications*, *18*(1), 60–80.

Yang, Z., Shangguan, L., Gu, W., Zhou, Z., Wu, C., & Liu, Y. (2014). Sherlock:

Micro-environment sensing for smartphones. *Parallel and Distributed Systems, IEEE Transactions on*, *25*(12), 3295–3305.

Ye, F., Ganti, R., Dimaghani, R., Grueneberg, K., & Calo, S. (2012). Meca: mobile edge capture and analysis middleware for social sensing applications. In *Proceedings of the 21st international conference on world wide web* (pp. 699–702).

Yoon, J. (2013). Three-tiered data mining for big data patterns of wireless sensor networks in medical and healthcare domains. In *Proceedings of the 8th international conference on internet and web applications and services, rome, italy* (pp. 23–28).

Yuan, B., & Herbert, J. (2014). A cloud-based mobile data analytics framework: Case study of activity recognition using smartphone. In *Mobile cloud computing, services, and engineering (mobilecloud), 2014 2nd ieee international conference on* (pp. 220–227).

Zaslavsky, A., Jayaraman, P. P., & Krishnaswamy, S. (2013). Sharelikescrowd: Mobile analytics for participatory sensing and crowd-sourcing applications. In *Data engineering workshops (icdew), 2013 ieee 29th international conference on* (pp. 128–135).

APPENDIX A: HETEROGENEITY IN MDSM APPLICATIONS

A.1 Bibliometric Analysis of WoS Databases

Research on mobile data mining has grown rapidly in recent years. A preliminary study of WoS databases was made by querying the string "mobile data mining". According to retrieved statistics, as of 28th January 2016, the WoS databases indexed 1930 publications in last 27 years (from 1990 to 28th January 2016) from International Scientific Indexing (ISI)-listed journals, conferences and workshop proceedings, and magazines (See Figure A.1). There was no significant research on the topic from 1990 to 2002. Since Year 2002, the miniaturization of technologies and onboard sensing technologies had geared-up the research on mobile data mining. However the major boom started from Year 2007 when both Google (*Android (Operating System)*, 2016) and Apple (*Apple IPhone History*, 2016) released their mobile phone operating systems.

According to Figure A.1, the number of publications rapidly increased till 2015 which shows that mobile data mining is continuously becoming a hot research topic. In

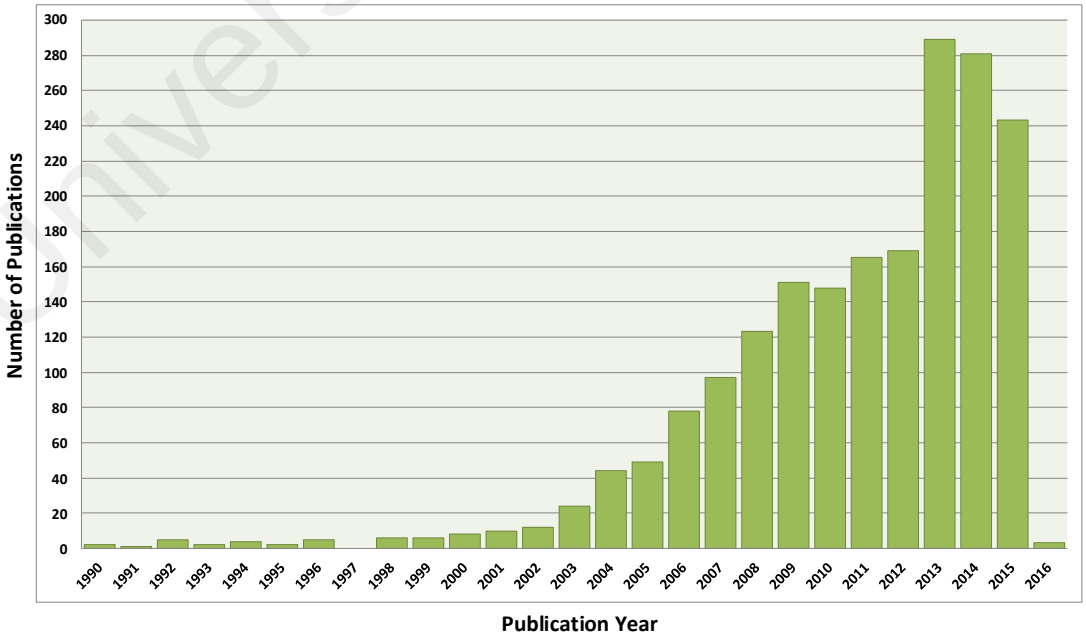


Figure A.1: Year-wise Publications (1992-2016).

near future we perceive a major shift towards the research on mobile data mining due to rapid growth in far-edge mobile devices for example smartphones, wearable devices, mobile Internet of Things (IoTs), and body sensor networks to name a few.

The citation trends for the topic "mobile data mining" are depicted in Figure A.2. The citation analysis showed that publications on the topic of mobile data mining obtained 9041 total citations from 8180 publications which were indexed in WoS databases. The popularity of research on mobile data mining is witnessed by the fact that 7935 citing publications were published without self-citations by the respective authors. The average citations per publication are 4.68 with h-index as 40. Figure A.2 also depicts that arrival of mobile operating systems in 2007, boomed the research on mobile data mining and it is still increasing day by day.

Since the main focus of this thesis is on mobile streaming data therefore we further analyzed the bibliographic records from WoS databases with another query string as "mobile data stream mining". We found 112 publications indexed by WoS databases from Year 1996 to 28th January, 2016. These 112 publications were cited by 343 other

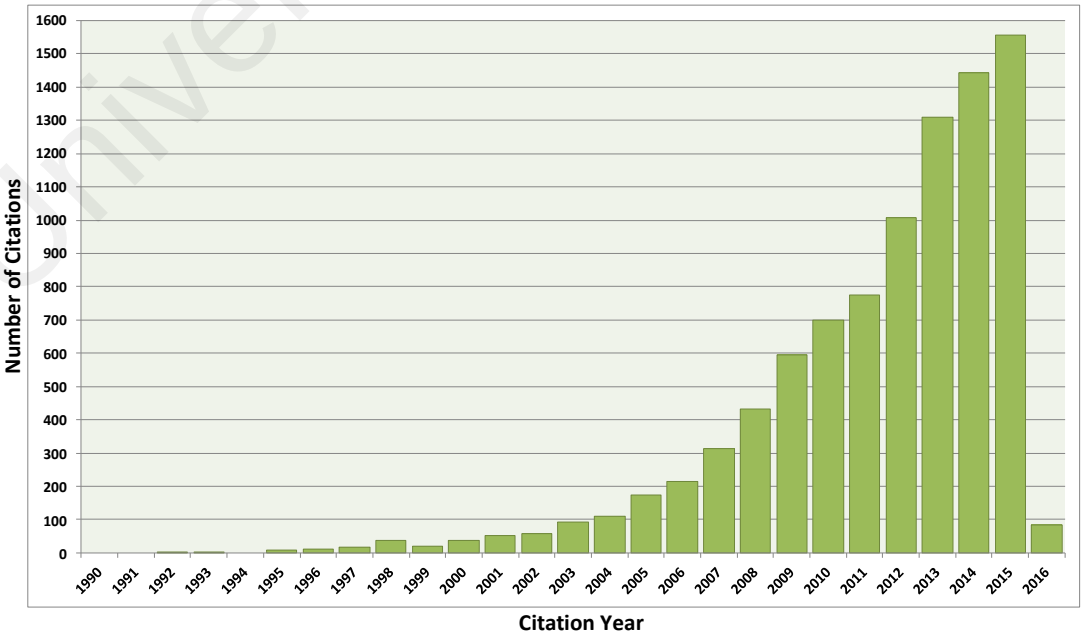


Figure A.2: Year-wise Citations (1992-2016).

publications in WoS databases whereby 331 publications do not contain any self-citation by respective authors. The average citations per publication are 3.06 with h-index as 11 which was lower when compared with bibliometric analysis of "mobile data mining" because less number of publications on the topic. Likewise, the major boom on "mobile data stream mining" was also witnessed after Year 2007 and it is rapidly growing.

A.2 Heterogeneity in MDSM Applications

The MDSM applications work in five steps: *a*) mobile applications provide functionality to acquire data streams from one or more data sources, *b*) fusion of data stream from multiple sources results in information rich data stream representing multiple facets of each data tuple, *c*) preprocessing operations enable to improve the quality of data stream by handling missing values, removing noise, and detecting anomalies and outliers, *d*) data stream mining operations are performed for online knowledge discovery using different model-based and model-less data mining algorithms, and *e*) uncovered knowledge patterns are summarized, integrated and managed for further utilization using multiple knowledge management approaches. Figure A.3 presents the taxonomy of heterogeneous MDSM applications.

A.2.1 Heterogeneity in Data Acquisition

Data acquisition in MDSM applications is a challenging task because of massive heterogeneity in multiple aspects.

Volume (Data Size): The MDSM applications need to handle continuous and unbounded data streams therefore limiting the size of data stream is a tedious task. MDSM applications handle volume using few methods based on sliding windows and segmentation (Krishnaswamy et al., 2012; Oneto et al., 2015; Abdallah et al., 2015; Wu et al., 2013). The sliding windows are used to sample preset number of tuples at a given time interval. The size of sliding windows may vary in different applications. Sliding windows

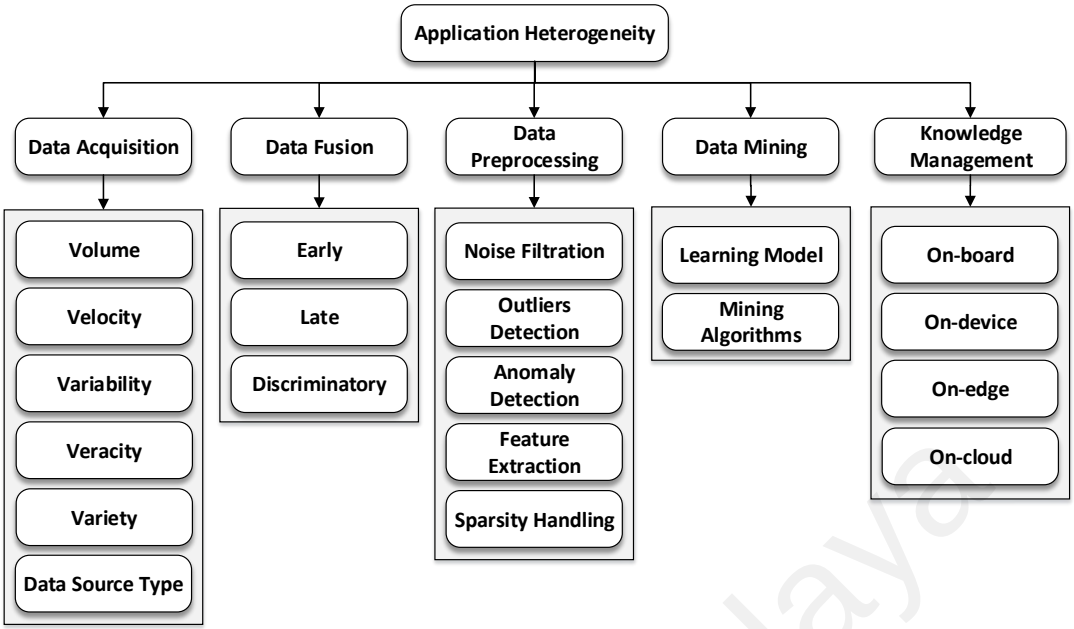


Figure A.3: Taxonomy of Heterogeneous MDSM Applications.

are used in two modes. The overlapping sliding windows contain a portion of data which overlap the previous window. The overlapping is performed to improve the quality of data stream so that the useful data items on the start and end of the windows should not be wasted. The non-overlapping sliding windows also play a vital role in some of the application areas. For example, mobile health applications with non-overlapping sliding windows are more useful than overlapping windows. Similarly data segmentation is used as an alternate of sliding windows methods where the buffered data streams are equally distributed in finite number of chunks for lateral processing.

Velocity (Speed or Data Rate): The speeds of incoming data streams play vital role in MDSM applications (Gaber, 2009). Velocity is the key challenge in mobile applications that increases makespan. MDSM applications handle velocity in two ways: *a*) raw data is collected in central data stores for lateral analysis and *b*) in-memory data analysis of data streams right after acquisition is performed. In the first approach, MDSM platforms create a delay between data acquisition and knowledge discovery. This strategy is more useful for analysis of historical data. The second approach is more appropriate for real-time data

analysis. In this case, MDSM applications need to compromise on the value of overall knowledge patterns due to one-time processing constraints of MDSM applications.

Variety (Number and Type of Data Sources): The heterogeneity of data sources and data formats in MDSM applications is represented by the variety property of streaming data (Rehman et al., 2015; Swan, 2012b). Data sources for mobile applications vary in terms of sensory and non-sensory data sources. In addition, the structured, semi-structured, and unstructured data formats also increase the variety in mobile applications. MDSM applications handle the variety challenge effectively to uncover maximum knowledge patterns.

Variability (Variable Data Production Rates:) Management of inconsistencies in mobile streaming data is attributed to variations. The variable data rates may cause computational overhead in peak-load times; therefore, a proper handling of variability property increases the usefulness of MDSM applications.

Veracity (Authenticity of Data Sources) The trustworthiness of mobile data streams is represented by veracity property (Swan, 2012a). This property is based on the authenticity of data sources and correctness of data. Effective handling of veracity property of data streams improves the overall effectiveness and usability of MDSM applications.

Data Source Type: The data sources in MDSM applications include sensory data sources such as accelerometer, camera, and microphone and non-sensory data sources such as application log files, device and communication interface scans etc.

Sensor Data Sources: Sensor configurations in far-edge mobile devices vary in terms of: *a*) locality, *b*) placement, and *c*) modality. The locality of sensors (Miller, 2012) is further referenced in two forms: *a*) sensors configured on the same motherboard as computational components, such as CPU, memory, and storage facilities; and *b*) sensors placed in remote wearable devices. Although useful for real-time data processing systems, local sensors-based far-edge mobile devices are disadvantaged by heat dissipa-

tion and energy consumption. Nonetheless, data collected from off-board sensors-based far-edge devices are prone to noise and communication protocol hurdles. Thus, deciding where to place sensor on body locations presents additional challenges for far-edge mobile device based data mining systems.

In MDSM applications, sensors can be configured in multiple ways (Miller, 2012): *a*) on-body through wearable devices; *b*) off-body through smartphones and other smart spaces; or *c*) sensors implanted inside the body. Selecting the best location for the sensors is critical in obtaining exact readings and noise-free data collection. For example, a study shows that far-edge mobile devices placed in the front and back of pants pockets produce different results (A. M. Khan, Siddiqi, & Lee, 2013). Therefore, sensor placement must be considered in any MDSM application.

Modality constraints in far-edge mobile devices (Augustyniak, Smoleń, Mikrut, & Kańtoch, 2014) can be categorized in two forms: *a*) to sense inside far-edge mobile device only; or *b*) to gather data from external environments as well. The fusion of data points from both modalities enable contextual information from external environments to be maintained. Consequently, different sensor configurations result in heterogeneity and complexity in data enabling feature-rich MDSM applications. A number of commonly known data sources in far-edge devices are presented in Table A.1. Notably, wearable devices generate sensor data streams, whereas smartphones may handle non-sensor data streams as well.

Non-Sensory Data Sources: In addition to sensor data streams, a huge variety of non-sensor data streams are present in far-edge mobile devices. These non-sensor data streams can be categorized as: *a*) device-resident; *b*) application-resident; or *c*) user-interaction based data. As a powerful far-edge mobile device, smartphones store a number of log files to maintain: *a*) communication and *b*) device-status related periodic information. Communication-related information includes Wi-Fi and Bluetooth scans as well as

Table A.1: Data Sources in MDSM Applications.

Type	Nature of Data Source	Data Source	Data Type
Sensors	Physiological	Heart Rate Monitor	Integer
		Blood Glucose Monitor	Integer
	Physical Activity	Accelerometer	Floating Point
	Environmental	Temperature	Floating Point
		Humidity	Integer
		Air Pressure Monitor	Floating Point
	Navigational	GPS Location	Floating Point
		Compass	Text
User Interaction	Input Data	On-screen Keyboard	Text/Numeric
		Microphone	Audio
		Camera	Image/Video
Device Resident	Application Logs	Web Browser Logs	Text
		Application-specific Logs	Text
	Communication Logs	Bluetooth Scans	Text
		Wi-Fi Scans	Text
	User Data	Contact List	Text
		Call Logs	Text
		SMS Data	Text

data about cellular networks and nearest cell towers. Status logs store information related to battery, operating system, and device hardware. Thus, a huge amount of data points can be gathered from device-resident information.

Mobile applications gather a variety of user-related information. For example, mobile web browsers maintain cookies to store user credentials and personal login information for social networks and mail-service providers. The sensitivity of this information suggests the need for privacy-preserving MDSM applications. Moreover, a user's interaction including the use of on-screen keyboards, microphone, and video cameras, is the key driver of non-sensor data production in far-edge mobile devices.

A.2.2 Heterogeneity in Data Fusion

Data sources generate data stream with different sampling frequencies and introduce heterogeneity in data fusion operations. For example, the sampling frequency of accelerometer is absolutely different when compared with a parallel data stream that is being sampled from Camera.

Early Data Fusion: Early data fusion methods are applied when raw sensor data from multiple data sources is sampled at the same sample rate which is measured as number of samples in each given time period (Oneto et al., 2015; Mukherji, Srinivasan, & Welbourne, 2014; A. M. Khan et al., 2013; L. Wang et al., 2012). For example, activity recognition applications sampling data streams from accelerometer and GPS location at the same time with same sampling rate. The average sampling frequency of accelerometer for activity recognition applications is recommended as 25Hz however user location do not change so frequently hence produce a lot of redundant GPS data. Similarly, if the sampling frequency is set as 1Hz the under-sampling of accelerometer produce inaccurate data hence affects the results of data mining algorithms. Therefore early data fusion strategies are helpful for MDSM applications with low sampling rates but under perform in case of high variance in sampling rates of different data sources.

Late Data Fusion: Late data fusion methods are applied after preprocessing the data stream (Min & Cho, 2011; Sherchan et al., 2012; Jayaraman, Gomes, et al., 2014). The late fusion strategies help in addressing the data redundancy issues. The data stream from multiple data sources is sampled at different sampling rates, preprocessed and the resultant data is integrated to generate an event data stream. For example, the accelerometer samples the sensors at 25Hz while the GPS is sampled at 1Hz. The late data fusion strategies first create sliding windows of 25 readings and performs the feature extraction from each sliding window. The extracted features and GPS locations are integrated and transformed into events. When compared with early data fusion, the late data fusion strategies helps in data reduction and improving data quality.

Discriminatory Features-based Data Fusion: Far-edge mobile devices such as wireless sensor networks and IoTs may involve homogeneous sensing settings where multiple data sources represent same information (Shoaib et al., 2014). However sensor configurations

and placement may affect in quality data acquisition. The discriminatory fusion methodologies involve the identification of quality data sources and fusion of discriminatory features which may help in improving the quality of uncovered knowledge patterns.

A.2.3 Heterogeneity in Data Preprocessing

The preprocessing operations enable to improve the quality of data stream. The heterogeneity in preprocessing operations arise when MDSM applications need to handle missing values, remove noise, and detect anomalies and outliers from the data stream.

Noise Filtration: Noise refers to the inclusion of extraneous and irrelevant information in mobile data streams (A. M. Khan, Lee, Lee, & Kim, 2010). The data streams becomes noisy due to multiple reasons such as improper placement of sensors, wrong sensor configurations, and inducement of environmental noise among others.

Outliers Detection: Outliers refer to misreported data streams where the acquired data streams do not fully represent the desired data streams (Hromic et al., 2015). A bundle of classification and clustering methods are used to detect and remove the outliers.

Anomaly Detection: Anomaly detection refers to the presence of anomalous data points in acquired data streams (Suarez-Tangil, Tapiador, Peris-Lopez, & Pastrana, 2015). The anomaly detection helps in improving the quality of knowledge patterns.

Feature Extraction: Massive data streams need to handle efficiently. The feature extraction methods help in extracting features (also known as attributes) from incoming data streams (Siirtola & Roning, 2013; Yang et al., 2014; Oshin, Poslad, & Zhang, 2015).

Sparsity Handling Highly sparse data may hamper the performance of far-edge mobile devices in some cases (Wickramasinghe & Ranasinghe, 2015). Similarly low sparsity degrades the performance of MDSM applications. Therefore, maintaining an adequate level of sparsity in MDSM applications help in improving the quality of knowledge patterns.

A.2.4 Heterogeneity in Data Stream Mining

Data stream mining algorithms vary in terms of frequent pattern mining, classification, and clustering schemes. Therefore, numerous types of supervised, unsupervised, and semi-supervised learning models are used for knowledge discovery in MDSM applications. The heterogeneity arises with type of data mining algorithms and their learning models.

Learning Model Heterogeneity: Machine learning algorithms, also called learning models (LM), play a significant role in MDSM applications. Although the selection and deployment of these models is difficult owing to onboard available resources, LMs are widely adopted in MDSM applications as well. LMs vary in terms of training type, training mode, and training modalities. The LMs are trained in three different ways.

1. **Supervised Learning (SL):** One of the most common tasks in data mining is to build models for the prediction of an object's class on the basis of its labeled attributes. A classification or regression model is usually trained for class prediction on large data sets. Model training is done using supervised learning, which allows for manual labeling of data points so that classification algorithms can predict similar unobserved data (Cord & Cunningham, 2008). An overview of SL algorithm development process is presented in Figure A.4. The designer first outlines the type and amount of training data that could help in building prediction models. The training data are acquired from multiple data sources, and feature extraction is performed for dimensionality reduction. In addition, instances are labeled manually (by user) or automatically (by application).

   The SL algorithms work by taking A (a set of input spaces with a_i feature vectors and L_i labeled attributes) and invoking a learning function that maps A to B (set of output spaces). The selection of function affects the overall performance of the

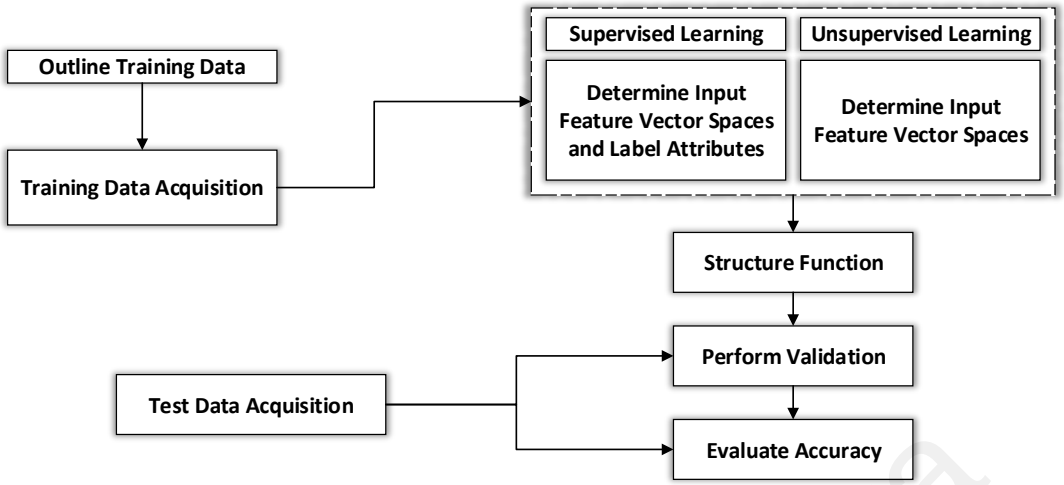


Figure A.4: Supervised and Unsupervised Learning Model Development Process.

SL algorithm. The evaluation of SL algorithm is performed using cross-validation, hold-out, prequential or leave-one-out techniques. Finally, the model is tested for accuracy using different evaluation criteria.

Formally, SL algorithm works under some assumption or bias for better predictions in unseen test environments. For example, smoothness assumption states that if two points, P_1 and P_2, are closer to each other in a training dataset, it is most likely that they will be closer in test data as well. In addition, an algorithm has high variance in SL settings if it predicts different output values when it is trained with different data sets; it is considered biased when it predicts correct results with systematically incorrect input spaces. The prediction error ($\hat{P}_e$) of a learned classifier is the sum of bias (β) and variance (σ^2) denoted as $\hat{P}_e = \beta + \sigma^2$. The trade-off between β and σ^2 is that learning model must be flexible with low β value so that it can fit the data well, but the high flexibility in learning model also increases σ^2 value. Therefore, a good SL algorithm provides a mechanism (automatic or manual) with which to adjust this trade-off between β and σ^2 and prevent over-fitting of the model (Cord & Cunningham, 2008; Dogan & Tanrikulu, 2013).

2. **Unsupervised Learning (UL):** The absence of class labels in data leads toward the discovery of new groups using UL techniques. Given the large data sets or streams with multiple attributes, the conventional SL algorithms require a great number of computational powers, making it difficult to manually handle all the grouping activities.

   The algorithm development process of UL methods, as depicted in Figure A.4, is the same except the labeling of data. In UL, the definition of training data and its acquisition is the same, but the input features vector space contains only unlabeled data. The learning model is trained and evaluated on the basis of input features vector space and finally tested using separate test data. The primary assumption for UL algorithms is that all data points are identically and independently distributed to define a $(n \times d)$ matrix. UL is initially used for density estimation, but now it is equally being adopted for outliers detection, clustering, quantile estimation, and dimensionality reduction (Huang, Song, Gupta, & Wu, 2014).

3. **Semi-supervised Learning (SSL):** The SSL plays an intermediate role between SL and UL methods. Both the scarcity of labeled data and the extensive labeling efforts are the main bottlenecks of SL algorithms. SSL extends SL by handling unlabeled data as well. Yet, SSL still needs human-intervention but reduces the effort of manually labeling the data. Moreover, SSL is equally exploited in other forms of data mining algorithms, such as clustering and regression. SSL algorithms work best under certain assumptions, and some of these known assumptions include smoothness assumption for classification algorithms, cluster assumption for interrelationship between cluster points, and low density separation assumption for dimensionality reduction algorithms. The assumptions with higher certainty level help develop better predictive models with higher accuracy. Alternately, poorly modeled assumptions can reduce the performance of predictive models.

The rest of the algorithm development process depicted in Figure A.5 is the same as SL and differs only in feedback propagation in learning models. The model is given positive feedback and is updated with labels of accurately predicted input vectors. Meanwhile, negative feedback is sent for re-training to training datasets. SSL algorithms work by first establishing a hypothesis from labeled data and then modifying or prioritizing the hypothesis using unlabeled data. For example, a SSL algorithm takes both A (labeled data) and B (unlabeled data) as input. Using A, it then builds a hypothesized model called $LM(A)$, then it processes B and based on the assumptions, it modifies or ranks $LM(A)$. The new model is called $LM(A+B)$, which means that it can handle both labeled and unlabeled input instances (Goldberg, Zhu, Singh, Xu, & Nowak, 2009).

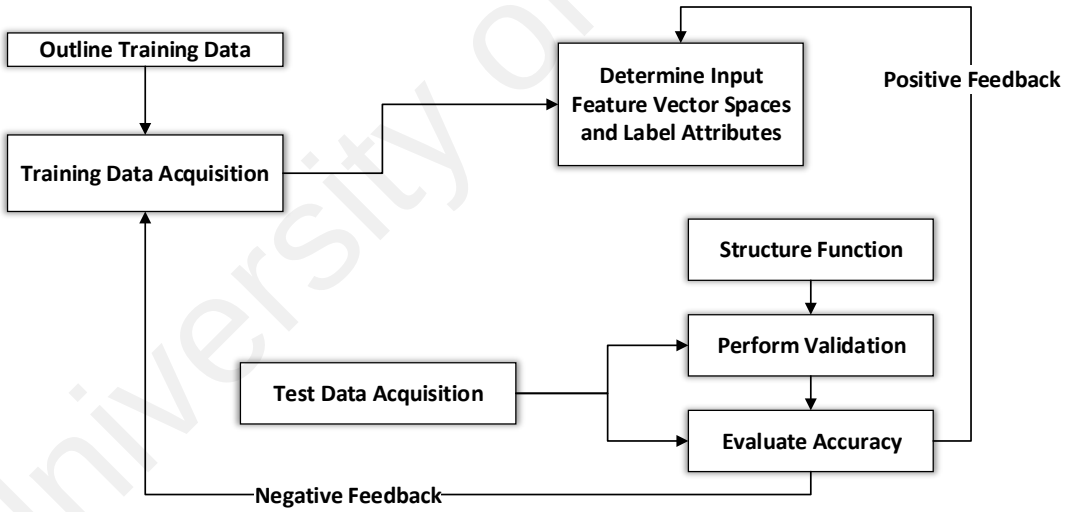


Figure A.5: Semi-supervised Learning Model Development Process.

SSL could be either transductive or inductive in nature. The transductive learning can only handle the known data points. Alternately, inductive learning enables one to handle unseen data as well. Some commonly used SSL methods include transductive SVM, co-training, self-training, and graph-based methods. An overview of existing literature reveals that there is no SSL method that can be categorized as the

best, but the authors (Triguero, García, & Herrera, 2015) recommends a checklist for method selection. They argued that expectation maximization with generative models may be a good choice for clustering algorithms. Similarly, co-training is appropriate for two set features, graph-based methods could be used for feature similarities, and self-training methods are best for complicated supervised classifiers. A detailed literature review of these methods can be found in (Triguero et al., 2015) for interested readers.

Training Mode (Online vs. offline): The LMs for MDSM applications vary in two dimensions (*i.e.*, off-line and on-line). In case of off-line learning, the learning models are first trained out-of-far-edge mobile devices and then used inside far-edge mobile devices, thus compromising accuracy as well as personalization at user-end (A. M. Khan et al., 2013; Liang, Zhou, Yu, & Guo, 2014). Alternately, on-line learning provides more accurate and personalized models, but less energy efficient because of computation intensities inside far-edge mobile devices(J. B. Gomes et al., 2012b).

Linear vs. Incremental Learning : Learning process takes place linearly or incrementally (Abdallah et al., 2015; Jayaraman, Gomes, et al., 2014; Min & Cho, 2011). Meanwhile, linear learning is more computation-intensive compared with incremental or ensemble learning. Hence, the choice of learning algorithm significantly affects the overall performance of MDSM applications.

Mining Algorithm Heterogeneity: The MDSM algorithms are categorized as classification, clustering, and association rule mining algorithms.

1. **Classification:** Three types of training models are used for classification in MDSM applications: *a*) universal, a single model that is used for all type of users; *b*) personalized, in which a model is trained for each individual; and *c*) adaptive, a model that starts with a universal scheme but gradually adapts and becomes personal for

each user (Lu et al., 2012). Each modeling technique, however, has some limitations. For example, the universal model is a single model for all users; therefore, the need for more accurate predictions arises because of differences in behavioral and physiological patterns of users. Personalized models are more accurate but require manual labeling for each activity. Finally, adaptive models must be self-trained and manually labeled at both times. Of the three, the universal modeling scheme is used in most of the studies in existing literature.

2. **Clustering:** Clustering using UL schemes creates multiple groups or clusters of highly similar or dissimilar data points (Abdallah et al., 2015; P. D. Haghighi et al., 2013; Suarez-Tangil et al., 2015). The assessment of such similarities and dissimilarities depends on the attribute values and distance measured from the cluster centroids. Different variants of data clustering techniques include hierarchical, spectral, subspace, and density-, centroid-, and constrained-based techniques. The choice of these techniques solely depends upon the type and nature of data to be clustered as well as the application requirements. However, clustering algorithms are not widely adopted in far-edge mobile device-based data stream mining systems due to high and sometimes unlimited computational requirements. In addition far-edge mobile devices face the challenges of dealing with concept drift and high dimensional noisy data streams in ubiquitous and pervasive environments.
3. **Frequent Pattern Mining:** Frequent pattern mining (FPM) is a key research area in knowledge discovery and data mining (Rehman et al., 2014a). Formally, FPM is basically applied over I(set of items):$\{i_1,...,i_n\}$ and T(set of transactions): $\{t_1,..,t_n\}$ where $T \subseteq I$ (Agrawal & Srikant, 1994; Rehman et al., 2014a). Transaction ID (TID) is used to uniquely identify a transaction in database. T contains A(a set of items) iff $A \subseteq T$. The association rule (AR): $A \Rightarrow B$ over two itemsets A and B exists iff $A \subset I$ and $B \subset I$ and $A \cap B = \emptyset$. The rule $A \Rightarrow B$ contains the transactions

with minimum support *minsup* for $A \Rightarrow B$ and confidence *minconf* for $A \cup B$. Moreover, for a given set of transactions D, the rule for minimum confidence (*minconf*) and minimum support (*minsup*) are specified by users and all rules that support *minconf* and *minsup* are generated for D resultantly. The itemsets and their ARs vary in simple, closed, maximal, rare, sporadic and utility based itemsets. Overall research in frequent pattern mining varies from basic patterns to multilevel and multidimensional patterns, to extended patterns for data sets and streams.

A.2.5 Heterogeneity in Knowledge Management

The integration, storage, and utilization of knowledge patterns in mobile data stream mining applications takes place at various places.

Onboard: The onboard storage refers to the storage capabilities of far-edge devices which is utilized to store locally uncovered knowledge patterns (N. Wang, Merrett, Maunder, & Rogers, 2013; Yoon, 2013). In addition, the synchronized knowledge patterns for personalized user experience are also stored onboard far-edge mobile devices.

on-edge: The knowledge management at edge servers help in achieving location aware distributed intelligence in MECC systems (Ye et al., 2012; Yoon, 2013). In addition, the location-aware aggregation of knowledge patterns facilitate in reduced data transfer in remote environments and minimize bandwidth utilization.

Remote: Conventionally knowledge patterns are integrated and stored in remote data stores which include cloud data center, servers, grid, and application server. Remote knowledge aggregation is useful for global knowledge discovery (Ferreira et al., 2010).

A.3 Handling Heterogeneity in MDSM Applications

This section presents the methods for handling heterogeneity at different levels in MDSM applications.

A.3.1 Methods for Handling Data Acquisition Heterogeneity

Number and type of data sources vary depending upon the nature of data stream mining systems. The application specific systems facilitate only essential data sources however the number of data sources in generic systems varies. For example, the application specific systems such as mobile activity recognition system mostly use accelerometers, GPS receivers, and magnetometers (Oneto et al., 2015; A. M. Khan et al., 2013; Yang et al., 2014). Alternately, generic systems like CARDAP, OMM, and MobiSens caters bundles of sensory and non-sensory data sources to enable generality and support wide range of applications (Jayaraman, Gomes, et al., 2014; P. D. Haghighi et al., 2013; Wu et al., 2013). The data sources include homogeneous and heterogeneous type of data sources. Homogeneous data sources are mainly used when same type of data is produced by multiple data sources such as multiple accelerometers deployed in wireless body sensor networks (Shoaib et al., 2014). Alternately heterogeneous data sources are used when MDSM applications need to collect and analyze data stream from different data sources. The heterogeneous data sources produce integrated and multi-dimensional data stream. Systems such as MineFleet and MobiSens utilizes heterogeneous data sources and produce multi-format information-rich data stream (Kargupta et al., 2010; Wu et al., 2013).

The data streams are collected from both onboard and off-board data sources. Similarly, the systems are designed as first-person data stream mining systems whereby personal data is analyzed and personalized knowledge discovery is performed (Gu et al., 2011; Mukherji et al., 2014). Alternately, the data stream mining systems integrate the data stream from multiple users/ devices/data sources for production of generalized knowledge patterns (Pasricha et al., 2015; Jayaraman, Perera, et al., 2014). MDSM applications handle multiple data types ranging from numerical and textual data to multimedia and event data streams. Literature review reveals that most of the systems cater only the

numerical data streams such as accelerometer axis and GPS coordinates however a few systems such as MSM (Mukherji et al., 2014) and OMM (P. D. Haghighi et al., 2013) supports multiple data formats. These data types finally lead towards the nature of data streams as structured, unstructured, and semi-structured data tuples.

To handle the resource constraints, MDSM applications adopt numerous data collection strategies which differ in terms of collection mode, and amount and nature of collected data. The data streams are either collected offline for lateral data processing or immediately processed using either onboard computational resource, offloaded to other computational system/infrastructures such as edge servers, cloud servers, or perform collaborative data processing by harnessing computational resources from nearer similar devices/systems. MDSM applications either collected raw data streams, or initially process and reduce the data streams to lower onboard resource consumption as well as bandwidth utilization cost for data offloading.

In addition, some studies reported the representative and context aware data collections strategies as well. The representative data collection strategies are useful when multiple data sources generate same data stream representing same knowledge. The representative data collection strategies works best in crowd-sensing like application scenario and useful in handling highly redundant data streams. The contextual information about user states, locations, and behavior helps in inferring current situations of users which in turn facilitate in data reduction whereby data stream mining applications only collect the data stream when specific situation occurs. For example, CAROMM utilizes context aware data collection strategies based on fuzzy situation inference model which infer current situation of users (Sherchan et al., 2012). Table A.2 presents the detailed literature review of methodologies for handling data acquisition used by selected studies.

Reference	Data Sources	Types	Name	Users	Data Types	Nature	Mode
(Oneto et al., 2015)	2	Off-board	Accelerometer and Magnetometer	Multiple	Numeric / Textual	Structured	Offline
(Pasricha et al., 2015)	1	Onboard	Application Log Files	NA	Textual	Structured	Online
(Abdallah et al., 2015)	72	Onboard	Accelerometer	Multiple	Numeric / Textual	Structured	Offline
(Suarez-Tangil et al., 2015)	Numerous	Onboard	Sequence of System Calls	NA	Textual	Structured	Online
(Boukhechba, Bouzouane, Bouchard, Gouin-Vallerand, & Giroux, 2015)	1	Onboard	GPS Receiver	Multiple	Textual	Structured	Offline
(P. D. Haghighi et al., 2013)	2	Off-board	ECG Sensors, Accelerometers	Single	Numerical	Structured	Offline
(J. B. Gomes et al., 2012a)	1	Onboard	Accelerometer	Multiple	Numerical	Structured	Online
(P. Liu et al., 2012)	1	Off-board	Accelerometers	Multiple	Numerical	Structured	Online
(A. M. Khan et al., 2010)	5	Onboard	Accelerometer	Multiple	Numerical	Structured	Offline
(Mukherji et al., 2014)	3	Onboard	Application Log Files, Call Records, Location	Single	Textual	Structured	Offline
(Abdallah, Gaber, Srinivasan, & Krishnaswamy, 2012)	72	Onboard	Accelerometer	Single	Numerical / Textual	Structured	Offline
(Sidek, Mai, & Khalil, 2014)	Numerous	Off-board	ECG Sensors	Multiple	Continuous Signals	Unstructured	Offline
(A. M. Khan et al., 2013)	5	Onboard	Accelerometer	Multiple	Numerical	Structured	Offline
(Srinivasan et al., 2014)	3	Onboard	Application Log Files, Call Records, Locations	Multiple	Textual	Structured	Offline
(Siirtola & Roning, 2013)	1	Onboard	Accelerometer	Multiple	Numerical	Structured	Offline
(Siirtola & Röning, 2012)	1	Onboard	Accelerometer	Multiple	Numerical	Structured	Offline
(Yang et al., 2014)	1	Onboard	Accelerometer	Multiple	Numerical	Structured	Offline
(Lu et al., 2012)	1	Onboard	Microphone	Multiple	Audio	Unstructured	Offline
(Donohoo, Ohlsen, Pasricha, Xiang, & Anderson, 2014)	Numerous	Onboard	GPS / user Interactions	Multiple	Numerical / Textual	Structured	Offline
(Oshin et al., 2015)	1	Onboard	Accelerometer	Multiple	Numerical	Structured	Offline

(Rai, Yan, Chakraborty, Wijaya, & Aberer, 2012)	1	Onboard	Accelerometer	Multiple	Numerical	Structured	Offline
(L. Wang et al., 2012)	7	Off-board	5 Accelerometers and 2 RFID	Multiple	Numerical	Structured	Offline
(Gaber, Stahl, & Gomes, 2014)	Numerous	Both	Multiple Data Sources	Multiple	Both	Both	Both
(Ortiz et al., 2015)	1	Onboard	Camera	Multiple	Images	Unstructured	Offline
(Braojos et al., 2014)	9	Both	Accelerometer	Multiple	Numerical	Structured	Offline
(Min & Cho, 2011)	Numerous	Both	Accelerometer and Magnetometer	Multiple	Numerical	Structured	Offline
(Stahl et al., 2012)	Numerous	Both	Multiple Data Sources	Multiple	Both	Both	Both
(Jayaraman, Perera, et al., 2014)	13	Both	Multiple	Multiple	Both	Both	Online
(Wu et al., 2013)	8	Onboard	Accelerometer, Magnetometer, Ambient Light, Temperature, Microphone, Camera, GPS, Wi-Fi	Multiple	Both	Both	Offline
(Sherchan et al., 2012)	Numerous	Both	Multiple Data Sources	Multiple	Both	Both	Offline
(Jayaraman, Gomes, et al., 2014)	Numerous	Onboard	Multiple Data Sources	Multiple	Both	Both	Online
(Lin et al., 2013)	1	Onboard	GPS Receiver	Multiple	Numerical	Structured	Offline
(Yuan & Herbert, 2014)	2	Both	Accelerometer and Gyroscope	Multiple	Numerical	Structured	Both
(Talia & Trunfio, 2010)	Numerous	NA	Numerous	Multiple	NA	NA	Offline
(Kargupta et al., 2010)	Numerous	Onboard	Onboard Vehicle Sensors	Multiple	Both	Both	Online
(Yoon, 2013)	2	Onboard	Accelerometer and GPS	Multiple	Numerical	Structured	Online
(Gu et al., 2011)	2	Off-board	Accelerometer and Camera	Multiple	Both	Both	Offline

Table A.2: Data Acquisition Heterogeneity.

A.3.2 Methods for Handling Data Fusion Heterogeneity

Literature review reveals that in most of the studies early data fusion is adopted whereby MDSM applications which collect data streams from multiple data sources and aggregate for further processing (Srinivasan et al., 2014; Braojos et al., 2014). Early data fusion results in redundant and noisy data streams therefore leads to inefficiency and extraneous resource consumption in mobile devices. A few studies use late data fusion strategies whereby collected data streams are preprocessed in parallel before data fusion (Min & Cho, 2011; Jayaraman, Gomes, et al., 2014). The late data fusion strategies consume onboard computational resources however it improves the data quality for lateral data processing. Late data fusion is useful when preprocessed data is integrated from multiple data sources. Although discriminatory data fusion strategies are also proposed by the researchers but existing literature still lacks its application in MDSM applications (Shoaib et al., 2014).

Data fusion strategies either work as online methods where all computations are performed in memory or works offline where data streams are stored onboard before data fusion (Oshin et al., 2015; Yoon, 2013). The online strategies are effective and improve system performance in terms of makespan and local storage I/O operations. However in-memory computations sometimes result in data loss and reduced data quality when dealing with large and complex data streams. Offline data fusion facilitates in improved data quality and complete data streams however quickly hampers onboard storage resources. Table A.3 presents a detailed literature review of data fusion heterogeneity in MDSM applications.

A.3.3 Methods for Handling Data Preprocessing Heterogeneity

The MDSM applications use various data preprocessing methods for sliding windows based data stream segmentations, feature extraction, data conversion from unstructured

Reference	Nature of Fused Data	Data Fusion	Fusion Mode	Fusion Objective
(Oneto et al., 2015)	Raw	Early	Offline	Linear acceleration and angular velocity
(Abdallah et al., 2015)	Raw	Early	Offline	Multi-sensor data acquisition
(P. D. Haghighi et al., 2013)	Raw	Early	Offline	Multi-sensor data acquisition
(A. M. Khan et al., 2010)	Raw	Early	Offline	To generate a linear feature space
(Mukherji et al., 2014)	Raw	Early	Offline	To produce a multi-dimensional dataset
(Abdallah et al., 2012)	Raw	Early	Offline	Multi-sensor data acquisition
(Sidek et al., 2014)	Preprocessed	Early	Offline	Acquisition of ECG signals
(A. M. Khan et al., 2013)	Raw	Early	Offline	To generate a non-linear feature space
(Srinivasan et al., 2014)	Raw	Early	Offline	To produce a high-dimensional dataset
(Donohoo et al., 2014)	Raw	Early	Offline	To generate feature vectors for context acquisition
(Oshin et al., 2015)	Raw	Early	Offline	Collection of data from multiple users
(Rai et al., 2012)	Raw	Both	Offline	To generate a quality feature space
(L. Wang et al., 2012)	Raw	Early	Offline	To generate a quality feature space
(Gaber, Stahl, & Gomes, 2014)	Raw	Both	Online	Knowledge aggregation
(Ortiz et al., 2015)	Raw	Early	Online	To generate a quality feature space
(Braojos et al., 2014)	Raw	Early	Online	To generate a quality feature space
(Min & Cho, 2011)	Preprocessed	Late	Online	Pattern integration and template matching
(Stahl et al., 2012)	Raw	Both	Online	Knowledge aggregation
(Jayaraman, Perera, et al., 2014)	Raw	Early	Online	Multi-sensor data acquisition
(Wu et al., 2013)	Raw	Early	Offline	Multi-sensor data acquisition
(Sherchan et al., 2012)	Preprocessed	Late	Offline	Collection of data from multiple users
(Jayaraman, Gomes, et al., 2014)	Raw	Late	Online	Collection of data from multiple users
(Lin et al., 2013)	Raw	Early	Online	Collection of data from multiple users
(Yuan & Herbert, 2014)	Raw	Early	Online	Collection of data from multiple users
(Talia & Trunfio, 2010)	Raw	Early	Offline	Collection of data from multiple users
(Kargupta et al., 2010)	Raw	Early	Online	Collection of data from multiple users
(Yoon, 2013)	Raw	Both	Online	Collection of data from multiple users
(Gu et al., 2011)	Raw	Early	Offline	Collection of data from multiple users

Table A.3: Data Fusion Heterogeneity.

to structured formats, signal analysis, noise and data filtration, addition of privacy and security features, dimension reduction, and outliers' detection, to name a few. The selection of preprocessing methods depends upon the nature of data streams and application requirements. For example, overlapping sliding windows based segmentations are used for activity recognition applications (Suarez-Tangil et al., 2015). Similarly, anonymization and encryption techniques facilitate in privacy and security features of MDSM applications (Mukherji et al., 2014).

Similar to data fusion operation, data preprocessing operations are performed in offline and online mode (Lu et al., 2012; Yuan & Herbert, 2014). The offline preprocessing methods are applied over historical data which is acquired and stored using onboard storage. The online data preprocessing operations are performed in memory. However in-memory computations become challenging due to variant complexities of data preprocessing algorithms. Table A.4 presents the detailed literature review of preprocessing methods.

Reference	Data Preprocessing Method	Type of Preprocessing Algorithm	Preprocessing Mode	Preprocessing Objective
(Oneto et al., 2015)	Sliding Windowing with 50% Overlap	Time and Frequency Domain Feature Extraction	Offline	Extraction of 561 Features
(Abdallah et al., 2015)	Clustering Sliding Windows	KNN Clustering	Online	Extraction of Features from Clusters and Sub-clusters
(Suarez-Tangil et al., 2015)	Sliding Windowing with 50% Overlap	Histogram Features	Online / Offline	Feature Extraction for Anomaly Detection
(P. D. Haghighi et al., 2013)	ECG signals converted using mobile health open source framework	ECG signals to numeric value conversion	Offline	Feature Extraction
(A. M. Khan et al., 2010)	Auto-regressive Co-efficient, Signal Magnitude Area, Linear Discriminant Analysis, and Kernal Discriminant Analysis	Noise Filtering and Feature Extraction	Offline	Feature Extraction from Non-linear Space
(Abdallah et al., 2012)	Clustering of Sliding Windows	Cluster-based Features	Online	Extracted Features from Clusters
(Mukherji et al., 2014)	Anonymization and Encryption	Privacy and Security	Offline	User De-identification
(Sidek et al., 2014)	QRS Selection / Normalization	Feature Selection and Normalization	Offline	Feature Extraction from ECG Data
(A. M. Khan et al., 2013)	Auto-regressive Co-efficient, SMA, LDA, and KDA	Noise Filtering and Feature Extraction	Offline	Feature Extraction from Non-linear Space
(Srinivasan et al., 2014)	Anonymization and Encryption	Privacy and Security	Offline	User De-identification
(Siirtola & Roning, 2013)	Feature Extraction	Statistical Feature Extraction Methods	Offline	Features Extraction from Accelerometer Data
(Siirtola & Röning, 2012)	Feature Extraction	Statistical Feature Extraction Methods	Offline	Features Extraction From Accelerometer Data
(Yang et al., 2014)	Feature Extraction	Time and Frequency Domain Feature Extraction	Offline	Feature Extraction
(Lu et al., 2012)	Feature Extraction	Statistical and Acoustic Features	Offline	Feature Extraction From Voice Data

(Donohoo et al., 2014)	Principle Component Analysis	Feature Extraction	Offline	Feature Extraction from GPS and other Onboard Sensors
(Oshin et al., 2015)	Feature Extraction	Mathematical Functions for Feature Extraction	Offline	Feature Extraction from Accelerometer Data
(Rai et al., 2012)	Feature Extraction	Statistical Feature for Time and Frequency Domain Features / Clustering of Higher-order Features	Offline	Feature Extraction
(L. Wang et al., 2012)	Dynamic Time Wrapping	Template Matching	Offline	Template Matching
(Gaber, Stahl, & Gomes, 2014)	Numerous	Numerous	Both	Multiple
(Ortiz et al., 2015)	Sift/Surb/ORB	Feature Extraction Method	Online	Feature Extraction
(Braojos et al., 2014)	Time-domain, Frequency domain, Wavelet	Feature Extraction Method	Online	Feature Extraction
(Min & Cho, 2011)	Segmentation	Activity based Classification	Offline	Segmentation
(Stahl et al., 2012)	Numerous	Numerous	Both	Multiple
(Jayaraman, Perera, et al., 2014)	Sliding Windows with 50% Overlap	FFT and Light-weight Analysis	Online	Multiple
(Wu et al., 2013)	Sliding Windowing for Segmentation	NA	NA	NA
(Sherchan et al., 2012)	Change Detection	Light-weight Clustering	Online	Quality Data Collection
(Jayaraman, Gomes, et al., 2014)	Light-weight Algorithms	Light-weight Clustering	Online	Quality Data Collection
(Yuan & Herbert, 2014)	Sliding Windowing with 50% Overlap and Feature Extraction	66 Time Domain and Frequency Domain Features Extracted through Statistical Methods	Online	Multi-user Data Collection
(Talia & Trunfio, 2010)	NA	NA	Offline	Multi-user Data Collection
(Yoon, 2013)	Filtration methods are applied	Filtration	Online	Data Filtration
(Gu et al., 2011)	Sliding Windows based Segmentation	NA	Offline	Improving Data Quality

Table A.4: Data Preprocessing Heterogeneity.

A.3.4 Methods for Handling Data Mining Heterogeneity

The MDSM applications use different learning models based on SL, UL, and SSL approaches. Currently most of the learning models are trained offline in desktop PCs, LAN servers, or CC systems. Some studies trained learning models in mobile devices as well however online training of learning models inside mobile environments is a challenging task. The challenge arises because training types of SL, UL, and SSL approaches differ. In case of SL models, the training data stream needs to be labeled/annotated so that learning models can accurately recognize and predict the future similar data streams. The labeling of data streams differs in manual, automatic, and observational settings. The manual labeling is performed when each segment/chunk of data stream is manually annotated however this process is quite laborious and needs a lot of efforts. An alternate methodology is the adoption of automatic application driven labeling where the applications are configured at the time of data collection and the resultant data streams are annotated accordingly. The automatic labeling is more promising as compared to manual labeling in order to reduce the training efforts. The observational settings further enhance the automatic labeling by allowing users to intervene in data labeling process. In this approach the learning models are initially trained in automatic settings however in case of discrepancies users are allowed to intervene by manually labeling the data streams.

The selection of learning algorithms significantly impacts the performance of MDSM applications in order to perform energy-efficient, cost-effective, highly accurate data stream mining operations. For deployment in mobile environments, the internal structures of learning models and their processing behavior play an important role in devising the computational complexity of learning models. In essence, MDSM applications need to perform online data stream mining operations on continuous data streams. Therefore most of the studies either separate the training and recognition processes or use shallow data

structures like arrays, lists, or pruned trees for improved efficiency.

Learning in MDSM applications is performed to achieve multiple objectives which include system level and application level performance enhancements. The system level performance objectives include battery life enhancements in mobile devices and performing offloading decisions in mobile cloud settings. However majority of the methods used learning models to enhance application performance in terms of change detection from uncertain data streams, model personalization, prediction and optimization of next locations, online activity recognition, finding emerging patterns, to name a few.

Once the learning models are trained and deployed, the MDSM applications process the incoming data streams in both online and offline mode. The offline data streams are first stored in onboard local storage and processed whenever the feasible environment for data stream processing is available. The online data streams are directly processed either using onboard computational resources or offloaded in other devices and systems in F2F, mobile-edge, MCC, or MECC settings. Majority of the studies in literature used classification algorithms due to low computational complexities and easy deployment as compared to clustering and frequent pattern mining algorithms. The classification algorithms are used for multiple purposes such as onboard classifications, on-wireless node classification, distributed classification, multi-level classification, and light-weight classification. A few studies implemented light-weight clustering and association rule mining algorithms which shows the practicality of clustering and frequent pattern mining algorithms in mobile-environments. Table A.5 presents a detailed literature review of data stream mining heterogeneity in MDSM applications.

Reference	Learning Mode	Learning Type	Learning Algorithm	Learning Objective	Mining Mode	Learning Model	Training Mode	Data Mining Algorithm	Purpose
(Oneto et al., 2015)	Offline	SL	Multiple Learning Algorithms	Battery Life Enhancement	Online	Yes	Offline	Feed Forward Selection	Classification
(Pasricha et al., 2015)	Online	SL	Q-learning	Battery Life Enhancement	Online	Yes	Offline/ Adaptive	Bayesian Classifier	Classification
(Abdallah et al., 2015)	Both	SL/ UL	Incremental/ Active Learning	Handling Concept Drift	Online/ Offline	Yes	Offline	Ensemble Classifier	Classification
(Suarez-Tangil et al., 2015)	Both	SL	ONB, J48, K-means	Offload or not to Offload	Online	Yes	Both	naive Bayes, J-48, K-means	Classification/ Clustering
(Boukhechba et al., 2015)	Online	SL	Habit Tree Data Structure	To Optimize and Predict Next Location	Online	Yes	Online	Association Rule Mining (ARM)	ARM
(P. D. Haghighi et al., 2013)	Online	SL/ UL	Multiple Learning Algorithms	Onboard Data Stream Mining	Online	Yes	Online	ARM, Classification, Clustering	Light-weight Data Mining
(J. B. Gomes et al., 2012a)	Online	SL	naive Bayes	Model Personalization	Online	Yes	Online	naive Bayes	Classification
(P. Liu et al., 2012)	NA	SL/ UL	Multiple Learning Algorithms	Proof of concept for Mobile Data Mining	Online	Yes	NA	Mobile WEKA library	Classification, Clustering, ARM
(A. M. Khan et al., 2010)	Offline	SL	Feed Forward Neural Network	Onboard Multi-sensor Activity Recognition	Online	Yes	Offline	Feed Forward Selection	Classification
(Mukherji et al., 2014)	Online	SL	Tree based	To Perform Context Prediction	Online	Yes	Online	Sequential Pattern Mining	Sequential Pattern Mining
(Abdallah et al., 2012)	Both	SL/ UL	Incremental/ Active Learning	Clustering	Online	Yes	Offline	K-means	Clustering

(Sidek et al., 2014)	Both	SL	NB, BN, and MLP	ECG Signal Classification for Biometric Identification	Online	Yes	Online	BN, MNN	Classification
(A. M. Khan et al., 2013)	Offline	SL	Feed Forward Neural Network	Onboard Multi-sensor Activity Recognition	Online	Yes	Offline	Feed Forward Selection Algorithm	Classification
(Srinivasan et al., 2014)	Online	SL	Tree based	To Perform Context Prediction	Online	Yes	Online	Sequential Pattern Mining	Sequential Pattern Mining
(Siirtola & Roning, 2013)	Offline	SL	Decision Tree and QDA	Model Training	Online	Yes	Offline	Decision Tree and QDA	Classification
(Yang et al., 2014)	Offline	SL	SVM	Model Training	Online	Yes	Offline	Support Vector Machine	Classification
(Lu et al., 2012)	Offline	SL	Gausian Mixture Model	Speaker Identification	Online	Yes	Offline	GMM, K-means, and EM	Classification
(Donohoo et al., 2014)	Offline	SL	LDA, LLR, SVM, VRL	Model Training	Online	Yes	Offline	LDA, LLR, SVM, KNN	Classification
(Oshin et al., 2015)	Offline	SL	EHMS	Model Training	Online	Yes	Online	EHMS	Classification
(Rai et al., 2012)	Online	SL/ UL	K-means and SVM	Higher Order Feature Extraction/ Model Training	Online	Yes	Online	Support Vector Machine	Classification
(L. Wang et al., 2012)	Offline	UL	Emerging Patterns	Activity Recognition	Offline	Yes	Offline	Emerging Patterns	Classification
(Gaber, Stahl, & Gomes, 2014)	Offline	SL	Hoefding Tree	Distributed Classification	Online	Yes	Offline	Emerging Patterns	Classification
(Ortiz et al., 2015)	Online	SL	NA	NA	Online	No	NA	K-means	Distributed Clustering
(Braojos et al., 2014)	Offline	SL	NFC	Classification	Online	No	Offline	Nero Fuzzy Classifier, DT	Multi-level Classification
(Min & Cho, 2011)	Offline	SL	SVM, NB, DT	Classification	Online	Yes	Offline	SVM, NB, DT	Multi-level Classification
(Stahl et al., 2012)	Offline	SL	Hoefding Tree	Distributed Classification	Online	Yes	Offline	Hoefding Tree	Distributed Classification
(Jayaraman, Perera, et al., 2014)	NA	NA	NA	NA	NA	NA	NA	Light-weight Algorithms	Classification, Clustering, ARM

(Dou et al., 2011)	NA	NA	NA	NA	Online	No	NA	K-means	Distributed Clustering
(Wu et al., 2013)	Offline	SL	EM	Classification	Offline	Yes	Offline	HMM	Classification
(Eom et al., 2015)	Online	SL	Instance based Learning, naive Bayes, Single Layer Perceptron	Classification	Online	Yes	Online	Instance based Learning, naive Bayes, Single Layer Perceptron	Machine Learning based Dynamic Task Scheduling
(Sherchan et al., 2012)	Online	UL	Light-weight Clustering	Change Detection	Online	Yes	Online	Light-weight Algorithms	Light-weight Data Mining
(Jayaraman, Gomes, et al., 2014)	Online	SL/UL	Light-weight Clustering and Classification	Local Analytics	Online	Yes	NA	Light-weight Algorithms	Light-weight Algorithms
(Lin et al., 2013)	NA	NA	NA	NA	online	No	NA	K-means	Clustering of GPS Data
(Yuan & Herbert, 2014)	Both	SL / UL	NB, DT, Nearest Neighbor, Neural Network	Universal and Personalized Model Development	Online/ Offline	Yes	Both	NB, DT, Nearest Neighbor, Neural Network	Personalized Activity Classification
(Hassan et al., 2015)	Online	SL	MLP, Linear Regression, SVM, Decision tree	Model Training for Predicting Offload-able Computations	Online	Yes	Online	MLP, LR, SVM, DT	Compute/Resource-intensive Methods Classification
(Talia & Trunfio, 2010)	Offline	NA	Numerous	NA	Offline	NA	Offline	Numerous	Multiple
(Kargupta et al., 2010)	NA	NA	NA	NA	Online	No	NA	Correlation and Distance Matrices Computations	Change Detection
(Yoon, 2013)	NA	NA	NA	NA	Online	No	Online	Multi-level Deployment of ARM Algorithms	ARM
(Gu et al., 2011)	Offline	UL	Emerging Patterns	Prediction of Emerging Patterns	Online	Yes	Offline	Emerging Patterns	ARM

Table A.5: Data Stream Mining Heterogeneity.

A.3.5 Methods for Handling Knowledge Management Heterogeneity

Since MDSM applications process data streams at multiple devices and systems therefore the integration and summarization of knowledge patterns needs careful attention. MDSM applications usually run the knowledge discovery operations such as learning and recognition and knowledge management operations such as integration, summarization, and storage of knowledge patterns at the same device or system. However, few studies present the synchronization/transfer of knowledge patterns among different systems whereby the knowledge patterns are stored either in local storage such as onboard data stores in far-edge mobile devices or in remote data stores such as those in cloud data centers and edge servers. The hierarchical knowledge management facilitate in enabling both local and remote storage settings. Hierarchical knowledge management strategies enable local storage at lower level where far-edge mobile devices mange the knowledge patterns using onboard settings. At the second level, multiple devices transfer the knowledge patterns to nearer edge servers which integrate and manage local data stores. Finally multiple edge servers in different geographical settings transfer the knowledge patterns to centralized cloud data centers which enable knowledge integration for a global view.

Knowledge visualization is another challenge that MDSM applications need to handle efficiently. MDSM applications provide the visualization functionalities either on-screen in far-edge mobile devices or provide a web interface for remote visualization. On-screen visualization in far-edge mobile devices is handy for real-time applications however limited screen size and energy intensive operations quickly hampers the onboard computational and battery resources. The knowledge management strategies work in both online and offline mode. The online knowledge strategies integrate, summarize and visualize the knowledge patterns before storage and lateral aggregation if required. However offline strategies first integrate, summarize, and store the knowledge patterns.

Reference	Local DM	Remote DM	On-screen Visualization	Remote Visualization	Knowledge Management
(Pasricha et al., 2015)	Y	N	Y	N	Online
(Abdallah et al., 2015)	Y	N	Y	N	Online
(Boukhechba et al., 2015)	Y	N	N	N	N
(P. D. Haghighi et al., 2013)	Y	Y	N	N	Online
(J. B. Gomes et al., 2012a)	Y	N	Y	N	Online
(Mukherji et al., 2014)	Y	N	N	N	N
(Abdallah et al., 2012)	Y	N	N	N	N
(Srinivasan et al., 2014)	Y	N	N	N	N
(Yang et al., 2014)	Y	N	Y	N	Online
(Gaber, Stahl, & Gomes, 2014)	Y	N	Y	N	Online
(Min & Cho, 2011)	Y	N	N	N	N
(Stahl et al., 2012)	Y	N	Y	N	Online
(Jayaraman, Perera, et al., 2014)	Y	Y	N	Y	Both
(Dou et al., 2011)	N	N	Y	N	N
(Wu et al., 2013)	N	Y	N	Y	Offline
(Sherchan et al., 2012)	Y	Y	Y	Y	Offline
(Jayaraman, Gomes, et al., 2014)	Y	Y	Y	Y	Offline
(Lin et al., 2013)	N	N	N	NA	Online
(Yuan & Herbert, 2014)	Y	N	N	NA	Both
(Talia & Trunfio, 2010)	N	Y	Y	N	Offline
(Kargupta et al., 2010)	Y	Y	Y	Y	Online
(Yoon, 2013)	Y	Y	NA	NA	Online

Table A.6: Knowledge Management Heterogeneity.

In such cases, the on-demand visualization is enabled whereby knowledge patterns are visualized when required. Table A.6 presents the detailed literature review of knowledge management heterogeneity in MDSM applications.

APPENDIX B: RULES FOR SCHEDULING

Rule_id	Rule	Action
r1	$F_{battery} \leq 10\%$	perform data collection.
r2	F != charging	device not charging, check battery capacity.
r3	F != locked	device not locked, check application status.
r4	UA_i == 'active'	application is running, perform data processing.
r5	$F_{battery} > 10\%$	perform data processing.
r6	F == charging	device charging, perform data processing.
r7	$M_{Battery}$> Required_AvgPower*0.5	on average, mobile edge server has sufficient battery power, start collaboration.
r8	$M_{Battery} \leq$ Required_AvgPower*0.5	on average, mobile edge server has insufficient battery power, look for other devices, or connect with Internet.
r9	F_{Memory}>Req_Memory	far-edge device has sufficient memory, process locally.
r10	$F_{Memory} \leq$Req_Memory	far-edge device has insufficient memory, process using mobile edge server or cloud services.
r11	$M_{Battery}$>Req_Battery*0.5	mobile edge server has sufficient battery for current process, offload data stream

		for collaborative execution.
r12	$M_{Battery}$ ≤Req_Battery*0.5	mobile edge server has insufficient battery for current process, offload data stream for cloud-based execution.
r13	M_{Memory}>Req_Memory*0.5	mobile edge server has sufficient memory for current process, offload data stream for collaborative execution.
r14	M_{Memory} ≤Req_Memory*0.5	mobile edge server has insufficient memory for current process, offload data stream for cloud-base execution.
r15	$UA_i \neq$ 'active'	MDSM application is active, start data processing
r16	UA_i == 'real-time'	MDSM application is real-time, must run data processing in either LA, CA, CLA, or low processing modes.
r17	$UA_i \neq$ 'real-time'	MDSM application is not real-time, check resources or perform data collection.
r18	Location == 'known'	location is known, perform data processing.
r19	Location == 'unknown	location is unknown, update list of locations.
r20	Connection == 'Wi-Fi'	connected with Wi-Fi, start data processing in CA or CLA.
r21	Connection ≠ Wi-Fi	not connected with Wi-Fi, stop

		data processing and continue data collection.
r22	Device == 'idle'	check application status, perform data processing.
r23	Device == 'busy'	stop data processing, continue data collection.
r24	$M_{memory} > Memory_{estimated}$	mobile edge server has sufficient memory to process current data stream, process in mobile edge server.
r25	$M_{memory} \leq Memory_{estimated}$	mobile edge server has insufficient memory to process current data stream, do not process in mobile edge server.
r26	$M_{Battery} > Memory_{estimated}$	mobile edge server has sufficient battery power to process current data stream, process in mobile edge server.
r27	$M_{Battery} \leq Memory_{estimated}$	mobile edge server has insufficient battery power to process current data stream, do not process in mobile edge server.
r28	$F_i.count > 0$	unprocessed files are present, start data processing.
r29	$F_i.count \leq 0$	no unprocessed file, stop data processing, continue data collection.
r30	Timer>0	timer is not expired, wait for result

		from mobile edge server.
r31	Mode==LA_{full}	start data processing in LA mode.
r32	Mode==$LA_{adaptive}$	disable all communication interfaces and inactive sensors and perform data processing.
r33	Mode==CA	connect through local wi-fi and perform data processing using mobile edge servers.
r34	Mode==CLA	connect through local wi-fi and perform data processing using cloud servers.
r35	M == known	mobile edge servers are known, perform data processing.
r36	M $\neq$ unknown	mobile edge servers are unknown, update list of servers and perform data processing.
r37	Timer$\leq$0	timer is expired, rescheduled F_i to other mobile edge servers or check Internet connection and upload in cloud.