UNIVERSITY OF CALIFORNIA,
IRVINE

Progression and Edge Intelligence Framework for IoT Systems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Zhenqiu Huang

2016

DEDICATION

To my parents, JianGuo Huang and YanHua Li.

TABLE OF CONTENTS

Page

LIST OF FIGURES **vi**

LIST OF TABLES **viii**

LIST OF ALGORITHMS **ix**

ACKNOWLEDGMENTS **x**

CURRICULUM VITAE **xi**

ABSTRACT OF THE DISSERTATION **xiii**

1 Introduction **1**
1.1 Background and Motivation 1
1.1.1 Service Oriented IoT 1
1.1.2 Edge Computing 2
1.1.3 Autonomic Computing 3
1.2 Contributions 4
1.2.1 Energy Aware Mapping 4
1.2.2 Scalable Deployment of IoT Applications 5
1.2.3 Edge Intelligence Support 6
1.2.4 Progression Framework 6
1.3 Dissertation Organization 7

2 Related Work **8**
2.1 Service Oriented Architecture 8
2.1.1 Service Composition and Selection 8
2.1.2 Service Oriented Middleware 10
2.2 WuKong Middleware Overview 10
2.3 Progressive System 14
2.3.1 Self Configuration 15
2.3.2 Self Optimization 17

3 Mapping with Energy and Location Consideration **21**
3.1 Energy Model and Constraints 22
3.1.1 Energy Model 23

3.1.2 Policy Defined Constraints 25
3.2 General Sensor Selection Problem in Multi-Hop Network 27
3.2.1 Quadratic Programming Formulation 27
3.2.2 Integer Programming Reduction 30
3.3 Communication Minimization Problem for Single-Hop Network 33
3.3.1 Problem Definition 33
3.3.2 Co-Location Graph 35
3.3.3 Complexity Analysis 36
3.4 Co-Location Graph and Selection 38
3.4.1 Layer Based Graph Construction 38
3.4.2 Selection Strategies 40
3.4.3 Co-Location Selection Framework 41
3.4.4 Mapping Remaining Services 43
3.5 Simulation Study 44
3.5.1 Simulation Setup 45
3.5.2 Performance Metrics 46
3.5.3 Performance Comparison 46
3.6 Summary of Energy Mapping 50

4 Mapping in Scalable Reconfiguration 51
4.1 Deployment Cost Model 53
4.1.1 Network Architecture 53
4.1.2 Reprogramming Latency Formulation 55
4.2 Sensor Mapping Without Considering Run-time Cost 57
4.2.1 Congestion Zone 58
4.2.2 Problem Definition 58
4.2.3 Computation Complexity 59
4.2.4 Integer Programming Formulation 61
4.3 Sensor Mapping Considering Run-time Cost 63
4.3.1 Run-time Latency Model and Constraint 63
4.3.2 Round-Up Algorithm Based on Linear Programming Relaxation . . . 67
4.3.3 Heuristic Algorithm for Relocating Services 67
4.4 Simulation of Scalable Deployment 69
4.4.1 Simulation Setup 69
4.4.2 Performance of Problem P_1 71
4.4.3 Performance of Problem P_2 73
4.5 Summary of Scalable Deployment 75

5 Edge Intelligence Support 76
5.1 Overview 76
5.2 Edge Server Implementation 77
5.2.1 Programming Model 78
5.2.2 Networking 79
5.2.3 Analytics Support on Streaming Data 80
5.3 Distributed Runtime of Edge Framework 85

5.3.1 Fault Tolerant Support 85
5.3.2 Online Model Retention 86
5.3.3 Model Checkpoint 87
5.3.4 Context Engine Integration 87
5.4 Edge Intelligence Case Study 87
5.4.1 EEG Control 88
5.5 Performance Study 95
5.5.1 Scalability 95
5.5.2 Fault Tolerant 99
5.5.3 Reconfiguration Overhead 100
5.5.4 Programming Simplicity 100
5.6 Summary of Edge Intelligence Support 101

6 Progressive System Design and Implementation 102
6.1 Overview 102
6.1.1 Attributes of Progression Model 103
6.1.2 Progression in System 104
6.1.3 Progression in Application 106
6.2 Progressive System Architecture 107
6.2.1 System Architecture 107
6.2.2 Progression Lifecycle 110
6.3 Progression Framework Implementation 112
6.3.1 Intercepter in Darjeeling 112
6.3.2 Progression Server 112
6.3.3 Optimality Test Support 113
6.3.4 Progression Services 114
6.4 Progressive System Case Study 118
6.4.1 Reconfiguration for Fault Tolerant 118
6.4.2 User Occupancy Aware Remapping 119
6.5 System Performance Study 120
6.5.1 System Setup 121
6.5.2 Remapping Latency 122
6.5.3 Reconfiguration Latency 123
6.5.4 Model Update Latency 124
6.5.5 Scalable Reprogramming Performance 125
6.6 Summary of Progressive System 127

7 Conclusions and Future Work 128

Bibliography 129

LIST OF FIGURES

Page

1.1 Foundation and Goal of Research 4
1.2 Organization of the Disseration 7

2.1 WuKong System Architecture 11
2.2 Flow Based Program Example 13
2.3 Distributed Progressive System Architecture 20

3.1 Flow Based Program for A Smart Home 22
3.2 Distance Aware Mapping Example 23
3.3 Co-location Mapping Example 35
3.4 Co-location Solution Comparison 43
3.5 Mapping Algorithms 44
3.6 T-Ratios for different selection algorithms. 46
3.7 Performance of different K sizes for linear FBP's. 47
3.8 T-Ratios for different FBP structures. 48

4.1 Reprogramming Latency Aware Mapping Example 52
4.2 System Model 53
4.3 Run-time Latency Example 61
4.4 Dominante Path Example 63
4.5 Device Setting Impact for Optimality 70
4.6 Gateway Scale Impact 71
4.7 Replica Number Impact 72
4.8 Deadline Miss Ratio Comparison 73
4.9 Reprogramming Latency Comparison 74

5.1 Edge Server Design 77
5.2 MPTN Architecture 80
5.3 Double Ring Buffer and Feature Extraction 81
5.4 Extendable Data Pipeline Structure 84
5.5 Edge Framework Run-time Overview 86
5.6 Delta Feature 91
5.7 Theta Feature 91
5.8 Alpha Feature 91
5.9 Beta Feature 91

5.10 Accuracy using Alpha and Beta 92
5.11 Accuracy using All 92
5.12 Parameter Gamma Impact 92
5.13 Parameter C Impact 92
5.14 Monitoring and Checkpoint Overhead 96
5.15 Localization Throughput 97
5.16 Scalability in Execution Time 97
5.17 Scalability in Response Time 98
5.18 Scalability in Memory Efficiency 99

6.1 Progression Model 103
6.2 Progressive Data Flow of Edge Intelligence 106
6.3 Progression Framework Architecture 108
6.4 Lifecyle of Progression Framework 111
6.5 Progression Server Design 113
6.6 Flow Based Program Reprogramming Data 115
6.7 System Setup for Performance Study 121
6.8 Remapping Latency 123
6.9 Component Replacement Latency 124
6.10 Model Update Latency 125
6.11 Reprogramming Algorithm Comparison 126

LIST OF TABLES

Page

3.1 Scalability with K = 4 and different (n, m) 49
3.2 Scalability with (n, m) = (50, 100) and different K 49
3.3 RF Energy Consumption per Bit 50

4.1 Time Distribution 75

5.1 EdgeClass Code Size 100

LIST OF ALGORITHMS

		Page
3.1	Co-Location Graph Construction	39
3.2	Selection Framework	42
4.1	Round-Up Algorithm for problem P_2	67
4.2	Service Relocation Algorithm	68
6.1	Replica Replacement Protocol	117
6.2	Occupancy Prediction Algorithm	119

ACKNOWLEDGMENTS

I would first and foremost like to express my deepest gratitude to my advisor Professor Kwei-Jay Lin for his guidance and support with my research work. His academic rigorousness, diligence and patience inspired me to do my best and to overcome all challenges towards becoming a qualified research.

I would like to appreciate my thesis committee members for all of their guidance through this processs; your discussion, idea, and feedback have been absolutely invaluable.

I'd like to thank my colleagues for both mentoring and assisting me along the way to my degree, including Jing Zhang, Sen Zhou, Weiran Nie, Yi-chin Chang, Jinhwan Lee. Their research motivate my ideas of research.

I would like to thank the following people who built Wukong at NTU as working base of my research, help my testing and harden the system, and give me feedabck: Bolung Tsai, Ching-Chi Chuang, Ginger Chou, Yu-Chung Wang, Chi-Sheng Shih.

I would like to thank Jingning He, Wei Huang, Sen Qian and Yue Zhang who are my intern mentors in Google and Uber. Due to their seriously guidance during the internship, I accumulated lots of engieering experience that later contributes to my research work.

I would like to thank visitoring scholars YingHua Sun, Jun Na and Yan. Their research in different areas broad my knowledge and also improve my capablity to tackle with research problems.

I would like to thank my dear friends Mengfan Tang, Hao Liu, QiGuang Wang, Hanyi Huang, May Huang, Tianyi Deng, Bin Wu, Chen Qin, Guantao Liu, Yingyi Bu, Wei Zhang, Zhijing Qin, for helping and enriching my life in Irvine and in the future.

I would like to appreciate all of brothers and sisters, special to Joseph Shie, Doris She, Jeff Ku, Steven Zhong, Oliva Fu, and Mary Fan, in the Church of Irvine who shepherding and nourish me in the spirit of Jesus Chirst. It is their ministration in love make the 6 year not only an experience of knowledge accumulation and exploration, but also a growth of spirit for me. No benefits goes out of Jesus himself.

Last but not least, I would like to thank my family for supporting me through all the years during my Ph.D. study.

CURRICULUM VITAE

Zhenqiu Huang

EDUCATION

Doctor of Philosophy in Electrical and Computer Engineering **2016**
University of California, Irvine *Irvine, California*

Master of Computer Science **2010**
Chinese Academy of Science *Beijing, China*

Bachelor of Computer Science **2006**
Northwestern Polytechnical University *Xi'an, China*

TEACHING EXPERIENCE

Teaching Assistant **2011–2015**
University of California, Irvine *Irvine, California*

PUBLICATIONS

1. Jun Na, Kwei-Jay Lin, Zhenqiu Huang, Sen Zhou. An Evolutionary Game Approach on IoT Service Selection for Balancing Device Energy Consumption, IEEE International Conference on e-Business Engineeering (ICEBE 2015), Beijing, China. Oct 2015

2. Zhenqiu Huang, Bo-Lung Tsai, Jyun-Jhe Chou, Chun-Yuan Chen, Chun-Han Chen, Ching-Chi Chuang, Kwei-Jay Lin, Chi-Sheng Shih. Context and user behavior aware intelligent home control using WuKong middleware, IEEE International Conference on Comsumer Electronics-Taiwan (ICCE-TW 2015), Taipei, Taiwan. June 2015

3. Zhenqiu Huang, Kwei-Jay Lin, Shih-Yuan Yu, Jane Yung-jen Hsu. Co-locating services in IoT systems to minimize the communication energy cost, Journal of Innovation in Digistal Ecosystems

4. Zhenqiu Huang, Kwei-Jay Lin, Shih-Yuan Yu, Jane Yung-jen Hsu. Building Energy Efficient Internet of Things by Co-Locating Services to Minimize Communication, ACM International Conference on Management of Energent Digital EcoSystems (MEDES 2014), Buraidah, Al Qassim, Saudi Arabi. Oct 2014

5. Zhenqiu Huang, Kwei-Jay Lin, Congmiao Li, Sen Zhou. Communication Energy Aware Sensor Selection in IoT Systems. IEEE International Conference on Internet of Things (iThings 2014), Taipei, Taiwan. Sept 2014

6. Shih-Yuan Yu, Chi-Sheng Shih, Jane Yuang-Jen Hsu, Zhenqiu Huang, Kwei-Jay Lin. QoS Oriented Sensor Selection in IoT System. IEEE International Conference on Internet of Things (iThings 2014), Taipei, Taiwan. Sept 2014

7. Zhenqiu Huang, Kwei-Jay Lin, Lina Han. An energy sentient methodology for sensor mapping and selectionin IoT systems. IEEE International Symposium on Industrial Electronics (ISIE 2014), Istanbul, Turkey. June 2014

8. Zhenqiu Huang, Kwei-Jay Lin, Jing Zhang, Weiran Nie. Performance Diagnosis for SOA on Hybrid Cloud Using the Markov Network Model. IEEE International Conference on Service Oriented Computing and Applications (SOCA 2013), Koloa, HI. Dec 2013

9. Yinghua Sun, Zhehui Wu, Zhenqiu Huang, Kwei-Kay Lin. A Computing Resource Market Model and Supply-Demand Matching Mechanism. IEEE International Conference on Cloud Computing and Big Data (CloadCom-Asia 2013), FuZhou, China, Dec 2013

10.Jinhwan Lee, Jing Zhang, Zhenqiu Huang, Kwei-Jay Lin. Context-Based Reputation Management for Service Composition and Reconfiguration, IEEE International Conference on Commerce and Enterprise Computing (CEC 2012), Hangzhou, China. September 2012

11.Jing Zhang, Zhenqiu Huang, Kwei-Jay Lin. A Hybrid Diagnosis Approach for QoS Management in Service-Oriented Architecture, IEEE International Conference on Web Services (ICWS 2012), Hawaii, USA. June 2012

12. Huanyu Ma, Jiang Wei, Songlin Hu, Zhenqiu Huang. Two-phase graph search algorithm for QoS-aware automatic service composition. IEEE International Conference On Service Oriented Computing and Applications (SOCA 2010), Perth, WA. Dec 2010

13. Wei Jiang, Charles Zhang, Zheqniu Huang, Mingwen Chen, Songlin Hu, Zhiyong Liu. QSynth: A Tool for QoS-aware Automatic Service Composition. IEEE International Conference on Web Service (ICWS 2010), Miami, FL. July 2010

14. Zhenqiu Huang, Wei Jiang, Songlin Hu, Zhiyong Liu. Effective Pruning Algorithm for QoS Aware Service Composition. IEEE International Conference on Commerce and Enterprise Computing (CEC 09), Vienna, Austria. July 2009

ABSTRACT OF THE DISSERTATION

Progression and Edge Intelligence Framework for IoT Systems

By

Zhenqiu Huang

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2016

Professor Kwei-Jay Lin, Chair

This thesis studies the issues of building and managing future Internet of Things (IoT) systems. IoT systems consist of distributed components with services for sensing, processing, and controlling through devices deployed in our living environment as part of the global cyber-physical ecosystem.

Systems with perpetually running IoT devices may use a lot of energy. One challenge is implementing good management policies for energy saving. In addition, a large scale of devices may be deployed in wide geographical areas through low bandwidth wireless communication networks. This brings the challenge of configuring a large number of duplicated applications with low latency in a scalable manner. Finally, intelligent IoT applications, such as occupancy prediction and activity recognition, depend on analyzing user and event patterns from historical data. In order to achieve real-time interaction between humans and things, reliable yet real-time analytic support should be included to leverage the interplay and complementary roles of edge and cloud computing.

In this dissertation, I address the above issues from the service oriented point of view. Service oriented architecture (SOA) provides the integration and management flexibility using the abstraction of services deployed on devices. We have designed the WuKong IoT middleware to facilitate connectivity, deployment, and run-time management of IoT applications.

For energy efficient mapping, this thesis presents an energy saving methodology for co-locating several services on the same physical device in order to reduce the computing and communication energy. In a multi-hop network, the service co-location problem is formulated as a quadratic programming problem. I propose a reduction method that reduces it to the integer programming problem. In a single hop network, the service co-location problem can be modeled as the Maximum Weighted Independent Set (MWIS) problem. I design algorithm to transform a service flow to a co-location graph. Then, known heuristic algorithms to find the maximum independent set, which is the basis for making service co-location decisions, are applied to the co-location graph.

For low latency scalable deployment, I propose a region-based hierarchical management structure. A congestion zone that covers multiple regions is identified. The problem of deploying a large number of copies of a flow-based program (FBP) in a congestion zone is modeled as a network traffic congestion problem. Then, the problem of mapping in a congestion zone is modeled as an Integer Quadratic Constrained Programming (IQCP) problem, which is proved to be a NP-hard problem. Given that, an approximation algorithm based on LP relaxation and an efficient service relocating heuristic algorithm are designed for reducing the computation complexity. For each congestion zone, the algorithm will perform global optimized mapping for multiple regions, and then request multiple deployment delegators for reprogramming individual devices.

Finally, with the growing adoption of IoT applications, dedicated and single-purpose devices are giving way to smart, adaptive devices with rich capabilities using a platform or API, collecting and analyzing data, and making their own decisions. To facilitate building intelligent applications in IoT, I have implemented the edge framework for supporting reliable streaming analytics on edge devices. In addition, a progression framework is built to achieve the self-management capability of applications in IoT. A progressive architecture and a programming paradigm for bridging the service oriented application with the power of big data on the cloud are defined in the framework. In this thesis, I present the detailed design of the progression framework, which incorporates the above features for building scalable management of IoT systems through a flexible middleware.

Chapter 1

Introduction

1.1 Background and Motivation

An emerging wave of Internet deployments, most notably the Internet of Things (IoT) systems, require management flexibility in addition to location awareness and low latency. Compared to original Wireless Sensor Networks (WSN), which are designed to operate at extremely low power, sensing the environment, simple processing, and forwarding data to the static sink in a uni-direction fashion, Internet of things systems deploy sensors in the environment for data collection and device control purposes in order to build smart applications such as smart homes, smart cares, and smart industries.

1.1.1 Service Oriented IoT

Internet of Things (IoT) systems deploy sensors in the environment for data collection and device control. Built on the foundation of wireless sensor networks (WSN), each IoT device can facilitate a set of sensors and actuators to connect with physical environments. Emerging

IoT systems often have heterogeneous sensors and can run multi-purpose applications, so that different applications may use a different subset of sensing devices according to their locations, capabilities and availability. As predicted by Gartner market research [3], there will be 50 billion IoT devices connected globally in 2020. The scalability of IoT system presents new challenges for device management with regard to energy efficiency, scalable application deployment for emergency situation and intelligence in application.

Firstly, heterogeneous devices are deployed in geologically distributed areas. This brings challenges for managing QoS of applications built on top of these devices. Secondly, with a changing environment, there comes the need for devices that are deployed with different purposes and capabilities. To collaborate with one another and to be sharable among different applications, devices will have tradeoffs between energy, run-time latency, reprogramming latency, and various kinds of quality of service (QoS) such as accuracy, latency in system deployment. These bring challenges to configuration of IoT applications and IoT devices.

1.1.2 Edge Computing

The rise of three important technologies: the cloud, smart devices, and mobile applications heralds a new age of remote intelligent nodes that not only communicate with each other but also analyze high volumes of local real-time data to make collaborative autonomous decisions. Dedicated, single-purpose devices are giving way to smart, adaptive devices that virtualize capabilities using a platform or API, collect and analyze data, and make their own decisions.

Forcasted by Cisco Global Cloud Index [1], the data created by IoT devices will reach 507.5 ZB per year by 2019. Such a large amount of data and requirement of time-sensitive applications will put massive pressure on the scalability of centralized data management systems on the cloud. To provide realtime processing of these data, the centralized data center solution will not only have one-time investment cost but also the cost on energy and natural resources, such as water for cooling. Advances in low-power computing and networking are pulling intelligence from the cloud to the edge of the network in the form of smart, connected devices. In the new edge computation model, smart devices, cloud services, and smart applications herald a new age of remote intelligent nodes that not only communicate with each other but also analyze high volumes of local real-time data to make collaborative autonomous decisions. Thus, an edge based IoT framework should devised to build intelligence in service oriented IoT systems.

1.1.3 Autonomic Computing

Autonomic computing is a concept that brings together many fields of computing with the purpose of creating computing systems that are able to be self-managing. Nevertheless, a centralized solution that manages geologically distributed large scale heterogeneous devices in the global IoT ecosystem is not effective and efficient. In order to achieve failure-resilience and self-management on Quality of Service (QoS) of heterogeneous devices, an evolving yet lightweight progression mechanism should be designed on the edge, so that these devices can reliably and progressively make autonomous decisions, and share resources and information with each other. At the same time, an edge intelligent application may be self-optimized by interplay with big data on cloud. To progressively manage system QoS and support intelligence, an extendable software framework is needed to manage runtime dynamics, such as the energy cost on device, end-to-end latency of application, and users' pattern and preference, etc. With such an extendable design, software module for single purpose may plug and play in the progressive framework.

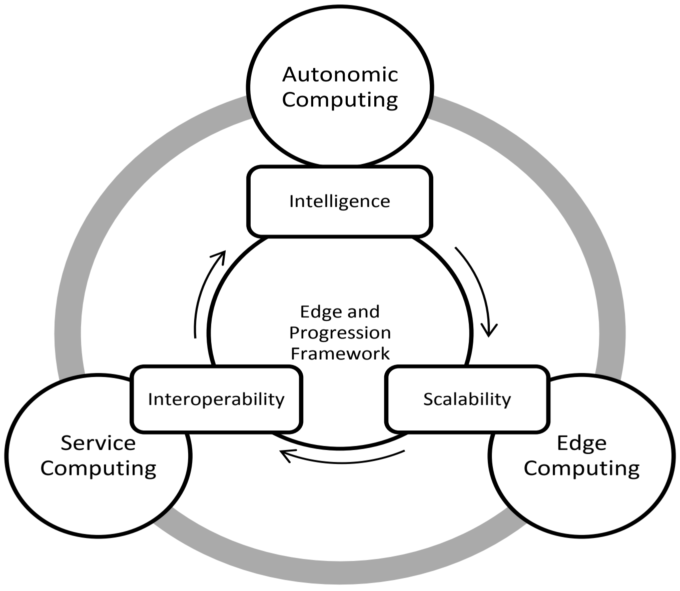


Figure 1.1: Foundation and Goal of Research

1.2 Contributions

The WuKong system [80, 81] is already built as a service oriented middleware of IoT systems. As shown in Figure 1.1, research in this disseration is to imrpove progressive and intelligence capability of WuKong by providing an scalable and extendable runtime management layer. I now summarize the contribution of the research reported in this dissertation.

1.2.1 Energy Aware Mapping

Ubiquitous sensing and actuating devices are now everywhere in our living environment as part of the global cyber-physical ecosystem. Sensing and actuating capabilities can be modeled as services to compose intelligent Internet of Things (IoT) applications. An issue for perpetually running and managing these IoT devices is the energy cost.One energy saving strategy is to co-locate several services on one device in order to reduce the computing and communication energy. I propose a service merging strategy for mapping and co-locating multiple services on devices. In a multi-hop network, the service co-location problem is formulated as a quadratic programming problem. I show a reduction method that reduces it to the integer programming problem. In a single hop network, the service co-location problem can be modeled as the Maximum Weighted Independent Set (MWIS) problem. I design an algorithm to transform a service flow to a co-location graph, then use known heuristic algorithms to find the maximum independent set which is the basis for making service co-location decisions. The performance of different co-location algorithms are evaluated by simulation in this thesis. Our simulation study shows that the MWIS algorithms can save 10% more communication energy than our previous solution, which is more than 30% energy than mapping result without any optimization.

1.2.2 Scalable Deployment of IoT Applications

Service Oriented Internet of Things (IoT) has been recently proposed for building smart environments as an integrated and scalable platform, where applications can be rapidly developed and deployed to adapt to context change. For some emergent situations, such as earthquake, an emergency handling application need to be proactively deployed to a large scale of regions with minimum latency. At the same time, the application instance in each region need to meet its end-to-end delay constraint. I study the sensor mapping design, which aims to minimize the reprogramming latency for large scale IoT systems while satisfying the run-time latency requirement at the application layer. I formally define it as the Integer Linear Programming (ILP) problem, and prove that it is NP-hard. An approximation algorithm based on LP relaxation and an efficient service relocating heuristic algorithm are designed for reducing time complexity. The performance of different mapping algorithms are evaluated by simulation in this study. Given a system with n congestion zones, our simulation study shows that our proposed solution can use only 1/n * 0.7 reprogramming time than a centralized solution to reprogram.

1.2.3 Edge Intelligence Support

As the era of Internet of things arrives, sensing devices are being used in many of our daily applications. Some desirable but not yet generally available capabilities include collaborative intelligence among heterogeneous devices, leveraging data analytics with low latency in a local environment, and adaptively controlling things according to a user's actual needs. I present the design of an edge framework which provides streaming capability in edge to make IoT smarter. I design an architecture of the edge framework that provides useful high-level primitives so that users can easily implement local data analytics on sensor and actuator streamings. My design simplifies the development of intelligent IoT devices. The edge framework has been built in the WuKong ecosystem. Moreover, I study the system performance for edge applications in a smart home environment. A Rasperberry Pi 2 device may parallel host more than 30 realtime EdgeObjects. Each the request to these EdgeObject may generates response within 1.5 seconds. Thus, these edge framework on edge device is fit for the intelligent applications in smart home and smart office in terms of throughput and parallelism.

1.2.4 Progression Framework

In the era of edge intelligence enabled IoT, large scale of devices are spread on the edges of network. This bring the challenge of efficient and effective management of such a widely geographically distributed system. The progression framework provides an scalable architecture and system support for building self-management, self-configuration and self-optimization capabilities for IoT application on the edge. I have built the system incorporated with the design of the edge framework, and studied the system performance in terms of monitoring, system reconfiguration and recovery time, and model self-tuning latency. In the design, the runtime management components, which are called PrClasses, are selected before FBP deployment and initialized through reprogramming. Such a software architecture provides great flexibility in supporting more policy driven runtime management capabilties for diversified system QoS requirement and users' preferences.

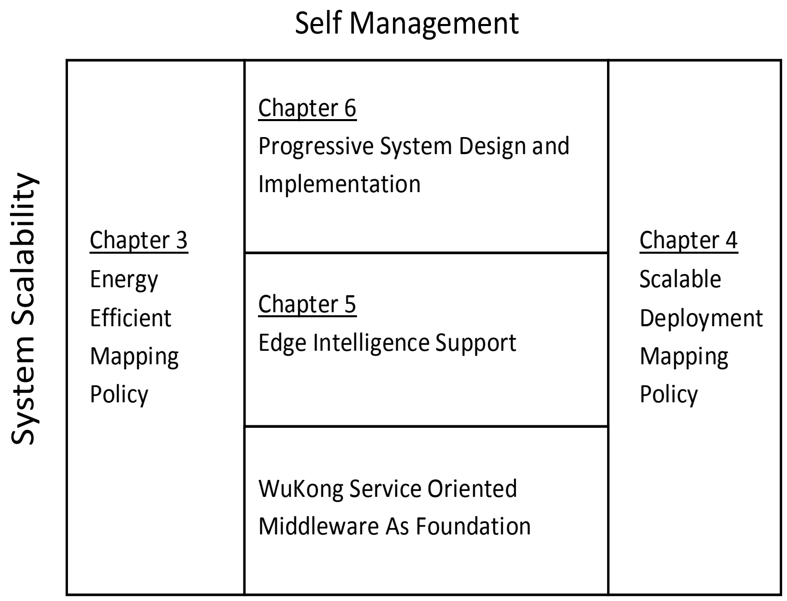

Self Management
System Scalability
Chapter 3
Energy Efficient Mapping Policy
Chapter 6
Progressive System Design and Implementation
Chapter 5
Edge Intelligence Support
WuKong Service Oriented Middleware As Foundation
Chapter 4
Scalable Deployment Mapping Policy

Figure 1.2: Organization of the Disseration

1.3 Dissertation Organization

The dissertation is organized as follows. Chapter 2 surveys the related work, and Chapter 3 introduces energy aware mapping in IoT. Chapter 4 presents the mapping strategy for application deployment in large scale regions, considering of both the reprogramming latency and run-time cost. Chapter 5 discuss the edge intelligence support that interplay of edge intelligence and big data on cloud in our edge framework. Chapter 6 shows the progressive system design and implementation that achieve diversified policy driven runtime management capability by using plug and play software architecture, followed by the conclusion remarks of my dissertation.

Chapter 2

Related Work

2.1 Service Oriented Architecture

Service oriented architecture (SOA) provides a powerful and flexible middleware paradigm for integrating distributed services into flow as an application. It promotes the idea of assembling services into a network of services that are loosely coupled to create fexible and dynamic service processes and agile applications that span organizations and computing platforms.

2.1.1 Service Composition and Selection

In service oriented systems, Quality of Service [65] (QoS) refers to various non-functional characteristics, such as response time, throughput, availability, and reliability. Services with a simliar and compatible functionality may be offered at different QoS levels. Thus, there may exist more than one way to build a composite service. QoS aware Service composition [12, 23, 78, 43, 51] and service selection [102, 99] have been the two most significant research problems and well studied in the service computing community. This research has investegated optimization methods that find the optimal service structure according to the user's multiple objected requirement.

More recently, the service composition problem has been specifically considered in applications, such as multimedia [33] and real-time systems [68, 50]. To tackle the issue of online rapid service composition, Alrifai [13] proposes the two phase selection method to reduce computation time while achieving close to optimal results. In the method, a coarse global optimization can be performed to decompose global QoS constraints into local constraints before using distributed local selection. To adapt to runtime dynamism of service, He et al [40] presents an approach to the adaptation of Web service composition based on work flow patterns. This approach measures the value of changed information (VOC) and the cost that updated services may potentially introduce in the business process. When the adaptation is expected to pay off, it will be performed within a certain scope defined by work flow patterns.

Instead of considering cloud or web service as computation resource in these above research, we treat sensor and actuator services in the context of the Internet of Things. In our research, sensor services run in embedded systems and are connected by low bandwith communication protocol, such Zwave, Xbee, etc. In such a communication resource limited environment, we mainly consider QoS, such as communication cost and communication latency, on the communication channel between services rather than the QoS of the service itself. In chapter 3, we study the service selection problem to minimize the communication energy cost in IoT. In chapter 4, we propose the hierchical service selection strategy to efficiently deploy large numbers of copies of an IoT application into a large IoT area through the communication resource constrainted network.

2.1.2 Service Oriented Middleware

Middleware as a software design concept combines common programming tasks into a reusable software layer. These tasks are usually related to distributed communication, for example to bring the familiarity of standalone programming to distributed programming and save application developers from the tedious and error-prone task of socket programming. Technologies that emerged for this purpose include DCE [66], SOAP [19], and REST [30]. Middleware continued to evolve to offer an abstraction layer over other types of heterogeneity such as programming language and operating system.

Service Oriented Middleware has leveraged the concept with the advent of the enterprise service bus (ESB), which has become valuable because it allows easy deployment of services via decoupling business logic with communication logic. Since service oriented architecture resolves the problem of scalability of internet based applications, it is used as the foundation of cloud computing, including paradigms such as Software as a service (SaaS), Infrastructure as a Service (IaaS) and Platform as a Service(PaaS). The scope of SOA adoption has not been limited to large IT enterprises, but also small and medium businesses, government sectors and health care providers, and cyber-physical systems. Aside from large computer servers, the technology has also been used in small and embedded devices such as sensor networks and mobile devices [87, 34, 64].

2.2 WuKong Middleware Overview

The WuKong project has built an intelligent middle-ware for IoT systems. The goal of the WuKong project is to help user design and develop hardware-independent IoT applications, so that they can be easily configured and dynamically deployed on vendor-independent IoT platforms.

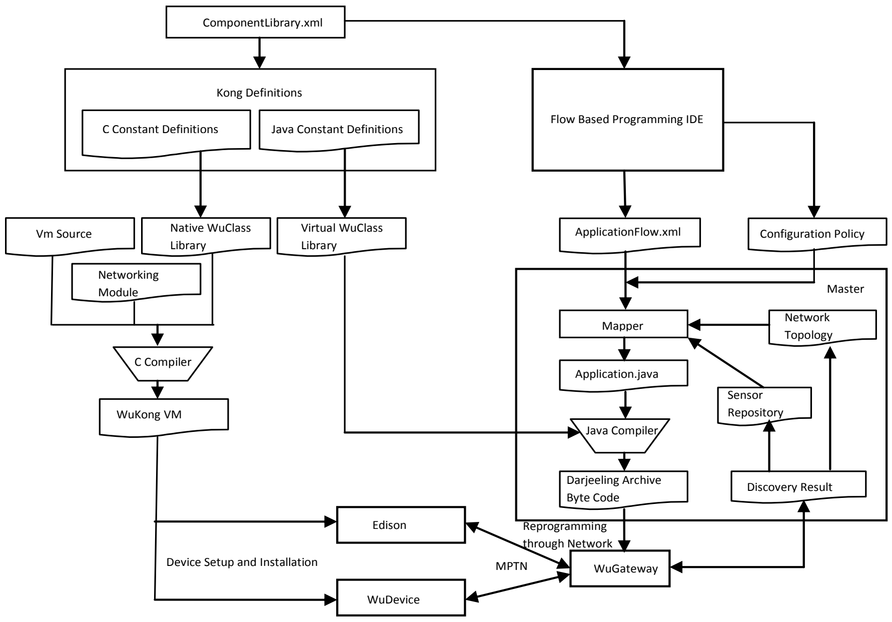


Figure 2.1: WuKong System Architecture

The WuKong system is implemented as a distributed computing run-time to accomplish requests from users and applications. Figure 2.1 shows the system architecture of WuKong middle-ware. WuKong system consist of the components on the cloud and that in deployment environment. In the deployment environment, WuKong deploys one WuKong master, several WuKong gateways, and number of WuKong devices. The functionality of each kind of devices are described below:

- WuMaster: WuMaster is short for WuKong Master. It is responsible for discovering, configuring, optimizing, and re-configuring sensors. To achieve this, it communicate with sensors through a layer of abstraction, hiding hardware and network details of the underlying sensor platform. During the discovery and identification phase, WuKong Master uses the profile framework to discover the capabilities of connected sensors, and configure sensors' parameters. It is also responsible for managing the user defined services in the system, including deployment FBP to devices, making in-situ decision for software upgrade and service remapping. In practice, the Master will be deployed on a computational powerful and robust server which is capable of reliably receiving user request and managing the services.

- WuGateway: WuGateway has two major responsibilities: communication gateway and backup master. As a communication gateway, it has the capability of discovering devices, forwarding messages, and dispatching message in heterogeneous networks. Communication gateway is named Multiple Protocol Transport Network (MPTN) gateway.
- WuDevice: WuKong devices. shorted as WuDevices, represent the networked physical devices in the system. One WuDevice can be a combination of sensors, actuators, and computation services. To be part of the WuKong systems, a WuDevice should register itself to WuKong master directly or via WuGateway, identify its own capability via its profiles, and join the system. WuKong VM is the virtual environment to execute the application logic. It consists of Darjeeling VM, networking module, and native profiles. Among these components, native profiles are architecture and platform dependent C WuClass library that interacts with physical sensors and actuators. The services including sensing, control, and computation carried out on WuDevices are deployed by master through remote programming.

The WuKong IoT middleware has the following major features:

- Virtualized IoT Devices: Virtualizing IoT devices allows hardware-independent application design and simplifies IoT services migration among devices without redefining applications. Consequently, one can deploy an IoT application on different hardware platforms without using hardware-and/or network-dependent application codes.
- Flow-Based Programming environment: WuKong provides a graphical flow-based programming (FBP) tool. In each FBP, a user can define the data and control flow to

build an IoT application. All the user has to do is to select the type of services from a predefined WuKong component library, drag and drop them on the programming canvas, and then connect them with directed links. The application flow diagram will then be used to map logical services onto appropriate physical devices during application deployment.

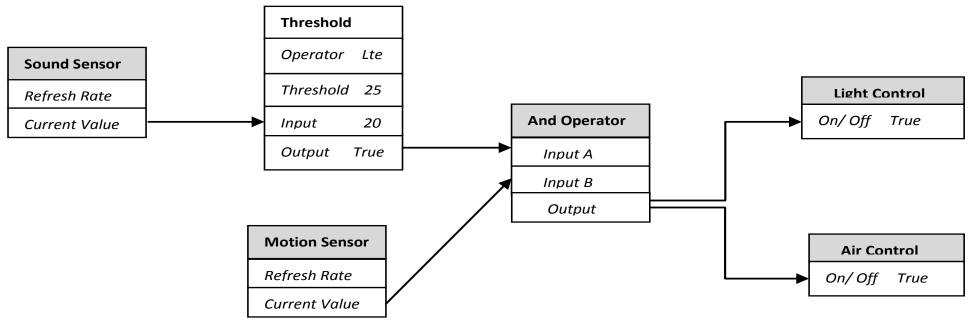


Figure 2.2: Flow Based Program Example

- Heterogeneous and Virtual services: The WuKong middle-ware provides virtual machines on heterogeneous platforms to simplify IoT application deployment and migration. Virtual IoT services can also be implemented (in Python) to support Web-based data services or UI running on servers, computers and smart phones. In addition, Darjeeling-based Java Virtual Machine is included in the WuKong middle-ware so that a system can dynamically add byte-code on each device.

- Deployment-time Service Mapping: To support heterogeneous and constant-evolving hardware platforms, WuKong delays the binding between logical IoT services and physical IoT devices until deployment. Consequently, during application development, platform-dependent properties and configurations such as port assignment and pin assignment minimized. The platform-dependent properties are collected by the WuKong

Master when a device registers itself in a WuKong system. The Master will use these properties to produce a proper configuration and generate required executable code for each IoT application.

One exmaple of a simple FBP is shown in Figure 2.2. It defines a simple indoor detection application which turns on light and air condition when system find there is a someone in the environment. In the FBP, the whole application is divided into 4 components, a PIR sensor, one sound sensor, a threshold, a light actuator, and an air condition actuator. Once detecting a signal from a moving object or people, the PIR sensor will send its signal to AND operator, which also receive signal from sound sensor if the sound value large than a threshold. If both of these two events happen, the light and air condition actuator will be triggered.

Given an application defined as FBP, WuKong middle-ware provides a set of mapping policy to optimize the deployment-time service mapping. In this dissertation, I mainly discuss the energy efficient mapping policy in chapter 3, which followed by the policy for scale mapping in chapter 3. The chapter 3 explores the problem of efficient mapping a large number of copies of a FBP into a large area through reprogramming. The goal of it is to minimize the deployment latency and meet the run-time execution deadline simultaneously. After that, I will introduce how to build intelligent application in edge framework, which is an extended capability in WuKong for streaming processing on edge. In the end, the progression framework for autonomic management will be present in chapter 6.

2.3 Progressive System

Autonamic computing is comprised of four aspects: self-configuration, self-optimization, self-healing, and self-protection. Among these four aspects, self-management and self-configuration are the essence of autonomic computing systems [53, 69]. We mainly consider the property of self-configuration and self-optimization of WuKong middle-ware in this disseration.

2.3.1 Self Configuration

In the autonomic computing definition, a self configuration system configure itself according to high-level goals, that is, by specifying what is desired, not nessesarily how to accomplish it. This can mean being able to install and set itself up based on the needs of the platform and the user. In WuKong middleware, we studied the mapping policies [44, 46, 61, 107] that select the optimal configuration for a predefined goal during application deployment. In this disseration, I mainly introduce the energy efficient policy and scalable deployment policy.

2.3.1.1 Energy Efficient Deployment

In the autonomic computing research community, power management is well studied in both data centers and wireless sensors networks(WSN). It has bean estimated that power equipment, cooling equipment, and electricity together are responsible for 63% of the total cost of ownership of the physical IT infrastructure of data center. It motivates researchers to optimize the power that a sevice or server will consume on a given infrastructure of a data center. The earlier study is mainly about individual server nodes' energy management[52] of the processer's power consumption, and then power models [55] that take memory, network, and IO consumption into account. In [29], power allocation of server clusters is considered.

In WSN, a common technique [95, 63, 96] to achieve energy efficiency is to put as many sensors in the sleep mode as possible, and keep only enough sensors in the active mode for sensing, communicating and processing. Wang et al. [91] propose a cross-layer sleep scheduling design in a service-oriented WSN while meeting the system requirement on the number of active service nodes for each service at any time interval. Another approach to prolong the network lifetime is energy consumption balancing. [73] studies the uneven energy depletion phenomenon in sink-based wireless sensor networks. [79] considers an energy efficient layout with a good coverage by using a multi-objective particle swarm optimization algorithm. [94, 24] propose two node deployment schemes, namely, distance-based and density-based, to balance each sensor node's energy consumption and to prolong network lifetime. In [45], we show how to use a quadratic programming model to balance the energy usage. In wireless sensor network research, earlier projects have focused on minimizing energy consumption on individual sensor nodes, whereas more recent studies have suggested that the energy efficiency for the whole system is actually more important for extending network lifetime [92]. Our research mainly consider the applications mainly on smart home and office, in which sleep schedule will increase the delay of communication and degrade of service usability. Therefore, we mainly consider two types of energy saving: one is changing mode of application, the other is minimization communication energy cost. To minimize communication energy, we focus on collocating services on devices, so that communication between two collocated serviced may be removed. The saving is small for a single communication link, but it is considerable in an environment with large scale deployment of IoT applications.

2.3.1.2 Efficient Reconfiguration

In wireless sensor networks, system reconfiguration is achieved by reprogramming devices. The reprogramming efficiency is always a major concern and has been well studied. Ealier research [86, 56] has focused on selectively choosing a set of sensors as code propagaters for reducing message collision in multiple hop wireless sensor networks. The sprinkler project [71] locally computes both a connected dominating set of devices to avoid redundant transmissions and a tranmission schedule to avoid collisions.

In our model, service oriented IoT applictions are composed of the WuClasses and WuObjects in embedded VMs, which are implemented to across architecture and platform. Designed specifically for WSN, UC Berkeley's Mate [57] was the first prove VM that can be made to run on sensor nodes. After that, serveral tiny JVM implementations comes out Darjeeling [22], Nano VM [4], and TakaTuka [15], which by providing only a subset of java, manage to shrink down to a size small enough to fit on sensor nodes. Wukong profiling framework is implemented on both NaonVM and Darjeeling, and integrated these two JVMs with communication support for Z-wave, Zigbee, and IP.

Compared to Ad-hoc networks in wireless sensor networks, the most recent IoT devices are connected by local gateways. At the same time, edge devices are usually placed near the source of data for real analytics and knowledge generation in the latest network architecture of IoT systems. Our work adopts the edge based system architecture and devises a decentralized reprogramming strategy that uses edge devices as reliable service selectors and code propagators. Thus, devices in different management zones can be parallel reprogrammed by different edge devices with low latency.

2.3.2 Self Optimization

Self-Optimization means a computing system that optimizes its use of resources. It may decide to initiate a change to the system proactively in an attempt to improve performance or quality of service. In the dissertation, we consider the self-optimization in both the application intelligence layer and the system layer.

2.3.2.1 Self Adaption of Application

Self-adaption is one of most important attributes of intelligence. The Internet of Things provides us with lots of sensor data. However, the data by themselves do not provide value unless turned into actionable, contextualized information. Big data and data visualization techniques allow us to gain new insights by batching-processing and offline analysis. Data centric self-adaption is widely used in search engine [20, 38, 39] and recommendation systems [10, 62], where both page rank score and utility matrix are continuously updated as more and more web page and user behavior logs are collected. The power that drives the adaptation is the distributed system for big data processing.

In open source communities, people have built a big data engineering ecosystem [74, 88] surrounding Map Reduce [26], which is a parallel programming paradigm devised by Google for processing big data. The life cycle of processing raw data to valuable knowledge in such a batching data pipeline, such as revert index processing at Google and social network analysis at Facebook, usually takes hours to days. To meet the real-time requirement of internet service, such as Uber for real-time ride-sharing service, the latency of data pipeline for real-time data dashboard and business decision making is reduced by Lambda architecture and the streaming system. Spark streaming [100, 101] developed the RDD memory data model, and a set of streaming transformation operator for batching processing with fault tolerant mechanisms. Another industrial streaming platform Samza [5]on YARN [90] use change-log stream to store the change log of internal nosql DB instances. Such a hybrid system may delivery valuable knowledge within minutes.

However, distributed intelligence systems can process far greater volumes of data at the edge than any centralized system. In fact, distributed systems scale horizontally: twice as much data can be processed at twice the cost, rather than the exponential cost curve of scaling a single system. In edge framework, we push the streaming processing capability to the edge. Within the edge server, we provide a set of feature extraction operators as a series of deterministic batch computations at small time intervals. At the same time, the checkpoint stream stored in pub/sub systems is devised to providing resilient support for the most valuable state in online learned models. By eliminating the need to get a response from a centralized management system, autonomous decision-making at the edge greatly increases responsiveness in time-sensitive applications. Nevertheless, our design may reduce the amount of data transmitted by individual devices and can lower network traffic, thereby reducing overall network latency. This is especially beneficial in environments where bandwidth is expensive or constrained, or where data must travel long distances.

2.3.2.2 Self Optimization of System

One of the earliest self-optimizing projects was initiated by DARPA for a military project called Situation Awareness System [6](SAS), which is to built adaptive routing enabled wireless network of mobile devices for soldiers to collect and transmit critical data on the battlefield. Another project called DASADA [2] was also initialized by DARPA to build mission critical systems that meet high assurance, dependability and adaptability requirement. After that, IBM proposed the concept of autonomic computing with self-X attributes in [42]. It defines a framework for how system will evolve to become more self-managing, and the key role to support autonomic behavior in heterogeneous system environment.

After that QoS management of Service Oriented Architecture (SOA) rises lots of attention. In the service oriented model, users and providers of services agree on a set of service level agreements (SLA). Thus self optimization in such systems is mainly formalized as resource allocation optimization problem [11, 70, 98, 97]. The Llama project [58, 60] resolve the accountability of service oriented middle-ware through agent selection [106], proactive diagnosis [104, 105] and service recovery [59, 103]. After that real-time attribute of Llama was also well studied [76, 77]. These studies focus on building adaptive management for one or multiple QoS in middle-ware. Our contribution in progression framework is mainly on defining a plug and play architecture on edge to support diverse QoS management for application in IoT middle-ware.

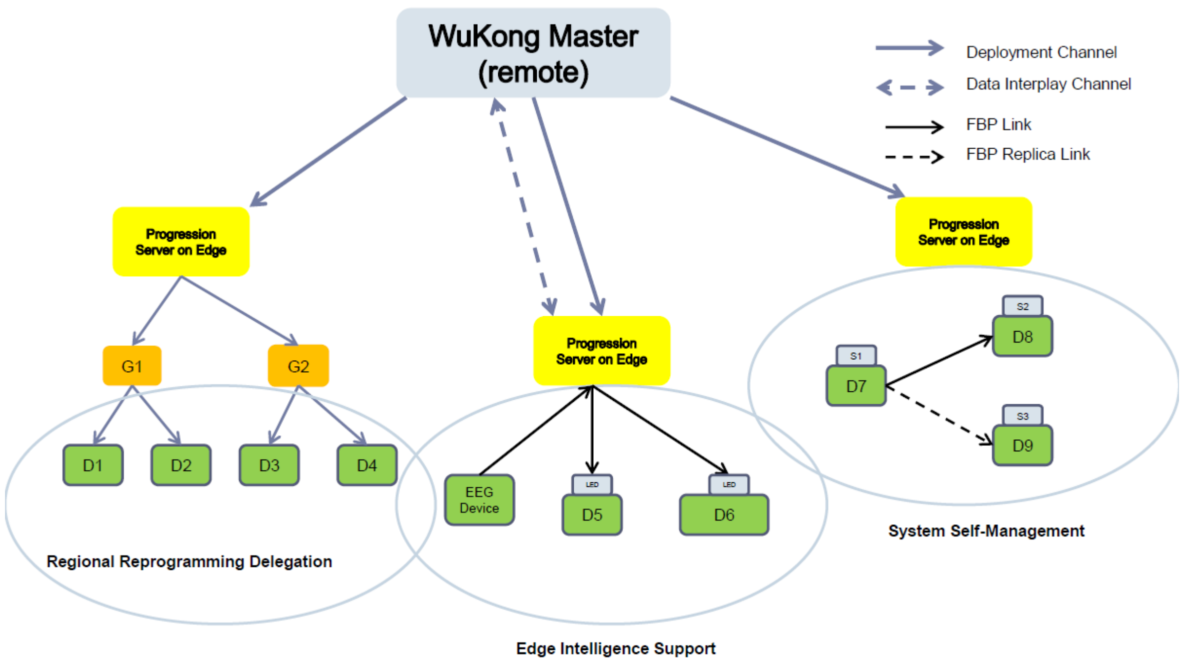


Figure 2.3: Distributed Progressive System Architecture

In IoT area, adaptive determinism of source providers of context and Ad-hoc Situation in home environment was studied in [49, 9]. A general QoS optimization framework and multi-agent based architecture was proposed for self-configuration and self-adaption in [16]. Our study proposes an optimization framework for mapping policies for WuKong applications on the edge. More recently, self-aware [72, 35] and self-healing [14] were studied in the context of smart buildings and cities. While our framework is to build an open scalable architecture that builds implementation foundation for these type of systems.

As shown in Figure 6.2, we propose a distributed progressive system architecture. In the architecture, the system reconfiguration through reprogramming, edge intelligence component, and local management are delegated by progression servers on edge. In such a progressive system, data are congested and processed locally. Thus, responsiveness of both systems and applications are improved by reducing the network traffic to remote master. The implementation of the distributed progressive system will be introduced in chapter6.

Chapter 3

Mapping with Energy and Location Consideration

One issue for perpetually running IoT services on distributed located devices is the energy cost. Running 50 billion devices and communicating among them will use a lot of energy. Researchers have proposed various device sleep scheduling algorithms [91] to keep some devices power off or running at a low-power mode. Another approach is to reduce network communication traffic to conserve energy. In this research, we investigate how to minimize the communication among devices. We use a service mapping scheme [44] to co-locate as many FBP services on the same device as possible to reduce distributed communication, in order to minimize the total energy cost for an application.

In our previous study [44], we use a simple greedy algorithm that co-locates two neighboring services with the largest communication cost first. In this paper, we present a comprehensive study on the service co-location problem for both multi-hop and single-hop networks. For devices in a multi-hop network, the service co-location problem is formulated as quadratic programming problem. We show a reduction method that reduces it to an integer programming problem. For single-hop networks, we present methods to find better solutions by transforming the service co-location problem to the Maximum Weighted Independent Set (MWIS) problem [82], which is a well-known data clustering problem. Using the MWIS model, we can find solutions that reduce about 10% communication energy from our previous solution in [44]. This paper is an extension of [47] by including theoretical complexity analysis as well as the study on multi-hop networks.

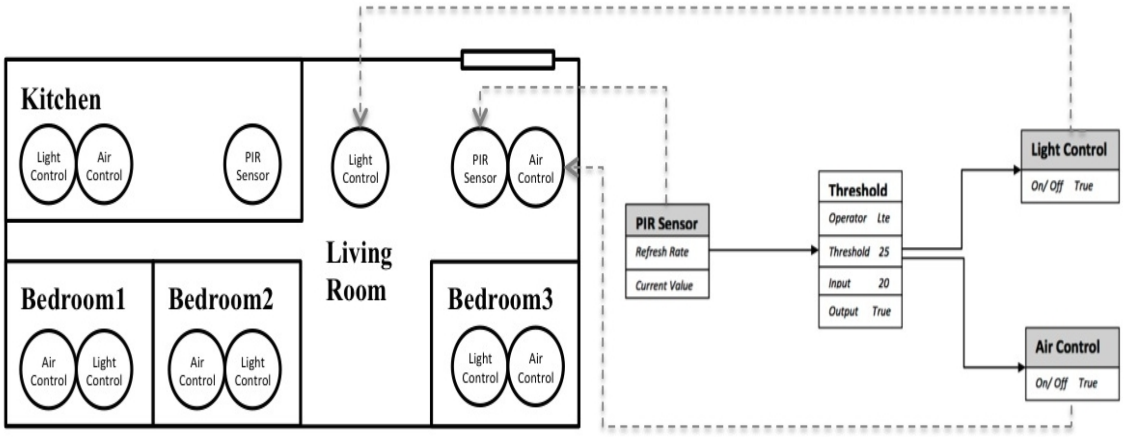


Figure 3.1: Flow Based Program for A Smart Home

3.1 Energy Model and Constraints

In our study, IoT systems are modeled as distributed systems with a set of sensor/computing/actuator devices that are placed on different locations in the target environment, connected by RF communication channels. As shown in Figure 4.1, each physical device D_i may host multiple sensors or computing components, called WuObjects, that are used to provide services. An application is composed by a network of virtual service components, each belonging to a service class (called WuClasses in WuKong) , denoted as C_i in Figure 4.1. Every WuObject can be used to fulfill a set of the WuClasses. For example, S_{11} and S_{21} on device D_1 can be used for C_1 and C_2 operations respectively.

In Wukong, an IoT application is defined by a network of virtual service *components*, each

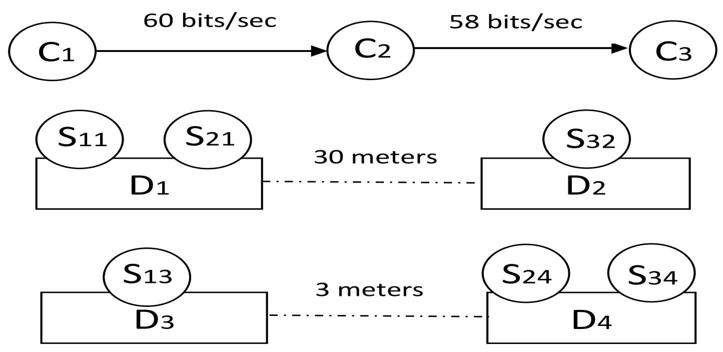


Figure 3.2: Distance Aware Mapping Example

of which belongs to a service class, called *WuClasses* in WuKong. Similar to the class definition in object-oriented programming, a WuClass defines the abstraction of functionality of sensing and actuating . WuKong supports the flow based programming (FBP) model so that application developers only need to define the flow of information between components. Each FBP is defined by a directed acyclic graph (DAG) $G(C, L)$ where C is a set of components C_i and L is the set of links $L_{ij} = (C_i, C_j)$ between components C_i and C_j. Using a GUI editor, users can define service processes using flow-based programs that collect readings from generic, or *virtual* sensors in a target environment, process data according to the decision logic, and generate desirable responses and actions in real time.

The WuKong middle-ware is used to support automatic device discovery, capability identification, and system configuration for FBP services defined by developers. WuKong is responsible for mapping FBP components to different physical devices with energy or location constraints.

3.1.1 Energy Model

WuKong device communication has been implemented using Z-Wave so that devices are directly reachable from each other by using one-hop communication. In [41], energy costs for transmitting and receiving k bits of data are formulated as follows:

$$E_T(k,d) = E_{elec} \times k + \epsilon_{amp} \times k \times d^2 \tag{3.1}$$

$$E_R(k,d) = E_{elec} \times k \tag{3.2}$$

where radio electronics parameter E_{elec} is about $50nJ/bit$ and transmit amplifier parameter ϵ_{amp} is about $10pJ/bit/m^2$. For a link L_{ij} between service components (C_i, C_j) that is mapped to communication between devices (D_x, D_y), the energy consumption U_{ij} of the link will be decided by the transmission energy t_{xy} on D_x and the receiving energy r_{xy} on D_y. But if a device hosts both end components of a link, then the communication energy cost U_{ij} becomes zero. Moreover, if we can map a link to a pair of devices with a small distance between them, the transmission energy t_{xy} will be reduced as well.

In Figure 3.2, we show two possible mapping decisions for FBP $\{C_1, C_2, C_3\}$, i.e. $\{D_1, D_2\}$ and $\{D_3, D_4\}$. In our earlier study [44], we use a layout parameter δ for calculating the the transmitting energy $E_T(k,d)$ as below:

$$E_T(k,d) = E_{elec} \times k \times (1+\delta) \tag{3.3}$$

The model only considers the data rate on links when make mapping decisions. Since the data rate on link L_{12} is the highest, the algorithm will co-locate C_1 and C_2 on D_1 to save the communication cost of link L_{12}. The energy cost of the whole FBP is 58 * (50 * 2 + 9)

= 6322 nJ/sec. In fact, we can find a better mapping option by taking the communication distance into consideration. Another option is to map component C_1 on device D_3 and co-locate components $\{C_2, C_3\}$ on device D_4. The energy cost of the second option is 60 * (50 * 2+ 0.09) = 6005.4 nJ/sec, which is smaller than the first option.

3.1.2 Policy Defined Constraints

In Wukong, application developers can use the policy framework to specify location and energy policies on mapping. We now show how we model policy definitions as constraints of the mapping problem.

3.1.2.1 Device Energy Constraint

Although energy harvesting technologies can be used to prolong IoT system's lifetime, only limited energy may be collected and charged within a period of time. Therefore, a user may specify a device energy consumption constraint e_k. Device energy constraints for the set of component S_D can be defined as below.

$$\mu(D_k) \leq E_k, \forall D_k \in S_D \tag{3.4}$$

3.1.2.2 Location Constraint

A user can specify a distance constraint R_{ij} between component C_i (e.g. temperature sensor) and landmark L_j (e.g. dining table). WuKong should check the locations of devices before deciding if a link can be used in the sensor co-location algorithm. During sensor selection, distance constraints for the set of component S_C should be imposed for the distance constraint R_{ij} as follows.

$$x_{ik} * dist(k, j) \leq R_{ij}, \forall C_i \in S_C \tag{3.5}$$

For example, in Figure 3.2, assume a user defines the device energy constraint on device D_2 as $E_2 = 1500$ nJ/second, and the location policy constraint $R_{34} = 25$m for the maximum distance from the device that hosts component C_3 to landmark L_4. Three constraints need to be added in the problem formulation.

$$x_{21}x_{32} * 58 * 30 + x_{24}x_{32} * 58 * 20 < 1500 \tag{3.6}$$

$$x_{32} * dist(2, 4) < 25 \tag{3.7}$$

$$x_{34} * dist(4, 4) < 25 \tag{3.8}$$

The energy constraint E_2 implies the first item $x_{21}x_{32}$ * 58 * 30 can't be realized. Therefore, the constraint filters out the mapping decision that deploys component C_2 on device D_1. In other words, by utilizing the ILP formulation on the device energy consumption and all the constraints, Wukong is able to find a mapping solution that have the longest system life-time as long as there is a feasible solution under these constraints.

3.2 General Sensor Selection Problem in Multi-Hop Network

In large IoT application scenarios like smart factory or smart building, an FBP will be mapped to devices whose communication to other devices may go through multiple hops. In this section, we present the energy model for multi-hop networks. We show how to formulate the problem as a quadratic programming problem, and how to solve it by integer programming.

3.2.1 Quadratic Programming Formulation

Given an FBP and an IoT system, we denote the data volume of an FBP link L_{ij} to be t_{ij} bits, the routing path between two devices D_n and D_m to be A_{nm} in the static routing table. During the mapping stage, the FBP link L_{ij} is mapped to a device pair (D_n, D_m), which are the start and end devices of the physical routing path A_{nm}. We define $H(L_{ij})$ to be the set of paths between all devices D_n and D_m where D_n can host C_i and D_m can host C_j. Let x_{ik}= 1 denote C_i has selected to use service S_{ik} on device D_k. Then, we can formulate the energy cost of $\mu(L_{ij})$ of link L_{ij} as:

$$\mu(L_{ij}) = \mu_T(L_{ij}) + \mu_R(L_{ij}) \tag{3.9}$$

$$\mu_T(L_{ij}) = \sum_{A_{nm} \in H(L_{ij})} x_{in} * x_{jm} * E_T(t_{ij}, D_n, D_m) \tag{3.10}$$

$$\mu_R(L_{ij}) = \sum_{A_{nm} \in H(L_{ij})} x_{in} * x_{jm} * E_R(t_{ij}, D_n, D_m) \tag{3.11}$$

i.e. the energy cost of L_{ij} is the cost to transmit and receive t_{ij} bits between devices D_n and D_m.

The energy cost $\mu(C_i)$ of a component C_i can be defined by:

$$\mu(C_i) = \sum_p \mu_T(L_{ip}) + \sum_q \mu_R(L_{qi}) \tag{3.12}$$

i.e. the energy cost of C_i is the sum of transmitting energy cost $\mu_T(L_{ip})$ of all its remote out-links L_{ip} in the FBP and receiving energy cost $\mu_R(L_{qi})$ of all its remote in-links L_{qi} in the FBP. The total energy consumption on device D_k can be defined by:

$$\mu(D_k) = \sum_i x_{ik} * \mu(C_i) \tag{3.13}$$

i.e. the summation parameter i represents the ith Wuclass on the device D_k.

The optimization objective function is to minimize the overall energy consumption among all nodes:

$$\min\,(\sum_k \mu(D_k)) \tag{3.14}$$

subject to:

$$\sum_k x_{ik} = 1, \forall 1 \leq i \leq N, 1 \leq k \leq M \tag{3.15}$$

where N is the number of components in an FBP, and M is the number of devices in an IoT system.

To show the problem is a quadratic programming problem, let us define the energy cost of component C_i on device D_k as $\mu^k(C_i) = x_{ik} * \mu(C_i)$. From Eq. 3.12, we have:

$$\mu^k(C_i) = x_{ik} * (\sum_p \mu_T(L_{ip}) + \sum_q \mu_R(L_{qi})) \tag{3.16}$$

Let us first expand the transmission unit using Eq. 3.10:

$$\mu_T^k(C_i) = x_{ik} * \sum_p \sum_{A_{nm} \in H(L_{ip})} x_{in} x_{pm} E_T(t_{ip}, D_n, D_m) \tag{3.17}$$

Since the constraints in Eq. 3.15 ensure that two devices D_k and D_n cannot be selected for deploying a particular component C_i at the same time, we can see that:

$$x_{ik} * x_{in} = 0, \quad k \neq n \tag{3.18}$$

Therefore, we can further simplify Eq. 3.17 as:

$$\mu_T^k(C_i) = \sum_p x_{ik}^2 \sum_{A_{km} \in H(L_{ip})} x_{pm} E_T(t_{ip}, D_k, D_m) \tag{3.19}$$

$$= \sum_p \sum_{A_{km} \in H(L_{ip})} x_{ik} x_{pm} E_T(t_{ip}, D_k, D_m) \tag{3.20}$$

In Eq. 3.19, since x_{ik} is a 0-1 integer variable, x_{ik} and x_{ik}^2 have the same value. Similarly, the receiving unit of Eq. 3.16 is:

$$\mu_R^k(C_i) = \sum_q \sum_{A_{nk} \in H(L_{qi})} x_{qn} x_{ik} E_R(t_{qi}, D_n, D_k) \tag{3.21}$$

Given the data volume of links and distances between devices are fixed in a problem instance, we can also calculate $E_T(t_{ip}, D_k, D_m)$ and $E_R(t_{qi}, D_n, D_k)$ as constants. In this way, we can define the energy consumption equation for each device. Thus, the final optimization problem is indeed a quadratic programming problem.

3.2.2 Integer Programming Reduction

Optimizing the quadratic programming problem is an NP-hard problem for which no polynomial algorithm is known. However, we can transform the problem to integer linear programming by rewriting the problem using new variable y_{ikpm} and constraints to take the value $x_{ik}*x_{pm}$ for every combination of x_{ik} and x_{pm}, and y_{qnik} for value $x_{qn}*x_{ik}$ also. The equivalence of two formulations has been proved in [18]. We thus obtain the following equivalent

0-1 linear programming definition:

$$\min\,(\sum_{k}\sum_{i}(\mu_T^k(C_i)+\mu_R^k(C_i))) \tag{3.22}$$

In Eq. 3.22, the transmission unit $\mu_T^k(C_i)$ and receiving unit $\mu_R^k(C_i)$ are thus transformed as follows:

$$\mu_T^k(C_i) = \sum_{p}\sum_{A_{km}\in H(L_{ip})} y_{ikpm}E_T(t_{ip}, D_k, D_m) \tag{3.23}$$

$$\mu_R^k(C_i) = \sum_{q}\sum_{A_{nk}\in H(L_{qi})} y_{qnik}E_R(t_{qi}, D_n, D_k) \tag{3.24}$$

In addition to the formal constraints defined, we need constraints for every y_{ikpm} and y_{qnik}:

$$y_{ikpm} \geq 0 \tag{3.25}$$

$$x_{ik} - y_{ikpm} \geq 0 \tag{3.26}$$

$$x_{pm} - y_{ikpm} \geq 0 \tag{3.27}$$

$$1 - x_{ik} - x_{pm} + y_{ikpm} \geq 0 \tag{3.28}$$

$$y_{qnik} \geq 0 \tag{3.29}$$

$$x_{qn} - y_{qnik} \geq 0 \tag{3.30}$$

$$x_{ik} - y_{qnik} \geq 0 \tag{3.31}$$

$$1 - x_{qn} - x_{ik} + y_{qnik} \geq 0 \tag{3.32}$$

These eight equations ensure the value of y_{ikpm} to be exactly the same as x_{ik} * x_{pm}, and value of y_{qnik} to be the same same as x_{qn} * x_{ik} in all cases. Therefore we can replace x_{ik} * x_{pm} with y_{ikpm}, and replace x_{qn} * x_{ik} with y_{qnik}. After we replace each x_{ik} * x_{pm} and x_{qn} * x_{ik} by its corresponding y_{ikpm} and y_{qnik}, the objective function is guaranteed to be optimal under the same setting of x_{ik} in the original objective function.

3.3 Communication Minimization Problem for Single-Hop Network

In smaller scale IoT systems, devices are installed close to each other and communicate with each other in a single hop network. In this section, we study the energy parameters for such systems. We present the problem definition and the analysis on the computation complexity of the problem.

3.3.1 Problem Definition

To study the problem complexity, we first formulate the mapping problem in a general problem $\boldsymbol{P_A}$. We then study a special class $\boldsymbol{P_K}$ of $\boldsymbol{P_A}$.

Given an FBP of n components to be edployed in a physical system of m sensing devices, the data communication of link L_{ij} between components C_i and C_j is known to be t_{ij} bits. The problem $\boldsymbol{P_A}$ is to find a mapping decision that maps each component in FBP to run on one device, while minimizing the total communication energy cost on these devices. In home environments, sensor devices may have a relatively uniform layout so that the distances between them are similar and do not make much difference on energy consumption. If so, we can simplify the second term of $E_T(t, d)$ in Eq. 3.1 to be independent of the distance between devices, and instead use a layout parameter, δ, i.e.

$$E_T(t, \delta) = E_{elec} \times t \times (1 + \delta) \tag{3.33}$$

With this approximated transmission energy model, we can define the transmission energy and receiving engery cost of link L_{ij} as below:

$$\mu_T(L_{ij}) = \sum_{A_{nm} \in H(L_{ij})} x_{in} * x_{jm} * E_T(t_{ij}, \delta) \tag{3.34}$$

$$\mu_R(L_{ij}) = \sum_{A_{nm} \in H(L_{ij})} x_{in} * x_{jm} * E_R(t_{ij}) \tag{3.35}$$

Then, we can define the energy cost $\mu(C_i)$ of a component C_i by:

$$\mu(C_i) = \sum_p \mu_T(L_{ip}) + \sum_q \mu_R(L_{qi}) \tag{3.36}$$

i.e. the energy cost of C_i is the sum of transmitting cost $\mu_T(L_{ip})$ of all its out-links L_{ip} in FBP and receiving cost $\mu_R(L_{qi})$ of all its in-links L_{qi} in FBP. Again we use variable $x_{ik} = 1$ to denote C_i has selected to run on device D_k. Then, we can find the energy consumption on a device as:

$$\mu(D_k) = \sum_i x_{ik} * \mu(C_i) \tag{3.37}$$

The objective function to minimize the total energy consumption on all devices is defined by:

$$\min\,(\sum_k \mu(D_k)) \tag{3.38}$$

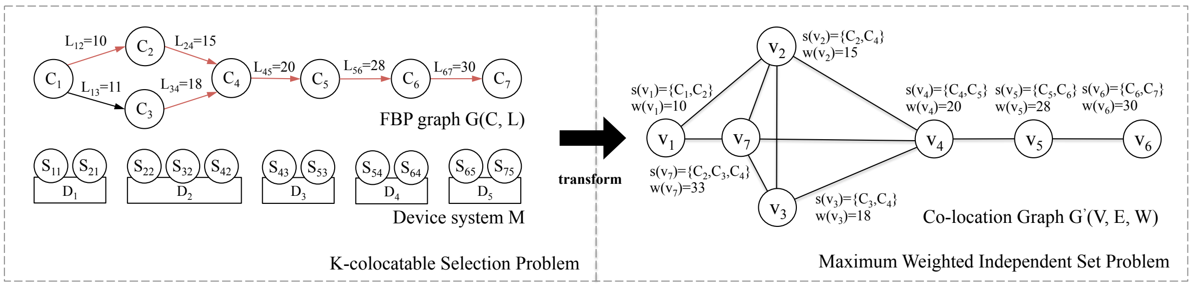


Figure 3.3: Co-location Mapping Example

subject to:

$$\sum_{k} x_{ik} = 1, \forall 1 \leq i \leq N, 1 \leq k \leq M \tag{3.39}$$

3.3.2 Co-Location Graph

We define problem $\boldsymbol{P_K}$ to be the K-sized co-location selection problem, if the total number of components that can be co-located on a device is no more than K in $\boldsymbol{P_A}$. The parameter K is determined by how many components exist on each device.

If we use the exhaustive search algorithm to solve the $\boldsymbol{P_K}$ problem, the time complexity is $O(mK * K^n)$, where n is the number of components in an FBP, m is the number of devices.However, some of them cannot be selected at the same time. For example, in Fig. 3.3, service component C_4 has 4 co-location options: $\{C_2, C_4\}$ on device D_2, $\{C_3, C_4\}$ on D_2, $\{C_4, C_5\}$ on device D_3, and $\{C_2, C_3, C_4\}$ together on D_2. These options are mutual exclusive since we can select only one device to deploy C_4.

We define a co-location graph as a vertex-weighted undirected graph $G(V, E, W)$, where V is a set of vertices that give all co-location options, E is a set of edges that represents the mutual exclusive relationship among co-location options, and W is a set of weighted labels that represent the gain when selecting a co-location option. Each vertex $v_i \in V$ represents a valid co-location option and contains a set of merge-able links. $s(v_i)$ is the set of WuClasses that may be co-located, and the weight $w(v_i)$ is the energy saving. The edge $e_{ij} \in G$ represents a conflict between v_i and v_j. For example, in Fig. 3.3, co-location option v_4 =$\{C_4, C_5\}$ and v_5 =$\{C_5, C_6\}$ are in conflict because they both have service C_5 which can reside on only one device, i.e. D_3 or D_4. The co-location node v_7 is for the option of placing C_2, C_3, C_4 together. In fact, a co-location graph may include vertices with many components co-located.

3.3.3 Complexity Analysis

We study the complexity of $\boldsymbol{P_K}$ by a reduction from the Maximum Weighted Independent Set (MWIS) problem to the co-location graph. MWIS is a well-studied graph problem [36, 82]. Let $G = (V, E, W)$ be a vertex-weighted undirected graph without loops and multiple edges, where V is the set of vertices, E is the set of edges, and W is the vertex weighting function. For any nonempty set S $\subseteq$ V, W(S) is defined by $\sum\limits_{u \in V} W(u)$. A subset I $\subseteq$ V is an *independent subset* of G if for any two vertices u,v $\in$ I, (u, v) $\notin$ E. An independent subset I of G is the *maximum* if there is no other independent subset I' of G such that $W(I) < W(I')$. MWIS is to find the independent subset from G that has the maximum total weight among all independent subsets.

If we find a MWIS solution for a co-location graph, the co-locations selected in those vertices will have no conflict with each other since they are not connected in the co-location graph. Moreover, the total weight is the maximum so that the energy saving is the largest.

Hastad [37] has shown that MWIS for a general graph is NP-hard in the strong sense. It is hard to approximate within $n^{1-\epsilon}$, for any $\epsilon > 0$. We now show the problem $\boldsymbol{P_K}$ is NP-hard even for K=2 by reducing it from the MWIS problem.

Theorem 1. *Problem $\boldsymbol{P_K}$ is NP-hard in the strong sense for $K = 2$.*

Proof. Given an instance of MWIS, we assume each vertex v_i contains two numbers p, q that represent component C_p and C_q, and no vertice has the same two numbers. Then, we can construct an instance of problem $\boldsymbol{P_K}$ with K = 2 as follows. We create a node $v^{'}$ for each pair of p and q, an edge $e^{'}_{pq}$ for node v_i, and put the weight $w(v_i)$ on $e^{'}_{pq}$ as the communication cost between $v^{'}_p$ and $v^{'}_q$. After that, we create a device D_{pq} with two components C_p and C_q for edge $e^{'}_{pq}$. Then, we use the constructed graph $G^{'}(V^{'}, E^{'}, W)$ as an FBP of $|V^{'}|$ components and a system with $|E^{'}|$ devices. Essentially, if we don't consider the weight, $G(V, E)$ is the line graph of $G^{'}(V^{'}, E^{'})$. Since every device only has 2 components, the problem constructed is a 2-colocatable problem.

Next, we show the two problem's optimal solutions are equivalent. Suppose that there is a maximum weighted independent set for the MWIS problem. We can use it to find the optimal co-location solution with K = 2. Using the optimal set for MWIS, if v_i is chosen, we co-locate two components C_p and C_q and place them on device D_{pq}. This decision saves the most communication energy consumption, implying the optimal solution for the objective function Eq. 3.38.

Similarly, if we have the optimal solution for a 2-colocatable problem, for the corresponding MWIS problem we can choose node $v^{'}$ of number p, q, if C_p and C_q of the FBP are co-located on device D_{pq} in the optimal solution for the 2-colocatable problem. The optimality of the solution for the 2-colocatable problem also ensures the optimality of the solution in the MWIS problem. $\square$

For P_K where $k \geq 2$, it can be transformed to an MWIS problem where some node contains more than 2 numbers. The above reasoning between an MWIS instance and the optimal solution for a K-colocatable problem still applies. Therefore P_K is NP-Hard for any K.

For each instance of P_A, we can find its upper bound on K by the maximum number of components in FBP that can be co-located together. In this way, we can conclude that the

P_A problem is NP-Hard.

3.4 Co-Location Graph and Selection

We are interested in finding efficient algorithms to solve the co-location problem. From the previous section, we can see that every mapping problem instance has a corresponding graph from which a MWIS could be used to solve the original co-location problem.

In this section, we first show how to construct a co-location graph $G_c = (V_c, E_c, W_c)$ from an FBP to be deployed in a system of IoT devices to depict all co-location options. In G_c, each $v_c \in V_c$ represents a co-location decision for a set of service components. An edge $(u_c, v_c) \in E_c$ represents a conflict between two neighboring co-location decisions.

After the co-location graph construction, we can solve the selection problem by using the MWIS algorithm on the co-location graph. We show three different greedy strategies of MWIS, define a greedy selection framework that could adopt these MWIS algorithms, and then show how we select devices for the remaining service components that have not been mapped yet.

3.4.1 Layer Based Graph Construction

We propose a general construction algorithm to include all co-locations in Algorithm 3.1. Since not all components connected by links in FBP are co-locatable, we first remove those non-candidate links and keep the list of co-locatable links in L. We also need the devices in the system as input M. M will be used by the algorithm to determine if there is a feasible co-location option by finding a device that can host those service components.

Algorithm 3.1 Co-Location Graph Construction

Input: A list of co-locatable links L as a connected graph and a system M

Output: Co-location graph $G(V, E, W)$

```
1: $X_k = \emptyset$, $1 \leq k \leq |FBP|$.
2: Set layer k = 1
3: Generate a vertex for each $L_{ij}$ in $L$ and add it to $G$ and $X_1$.
4: while $X_k \neq \emptyset$ do
5:     for all co-location vertex $v_i \in X_k$ do
6:         for all $v_i$'s generator $v_j \in g(v_i)$ do
7:             add edge $e_{ik} = (v_i, v_k)$ to $G$, for every $v_k$ that is $v_j$'s neighbor
8:         end for
9:     end for
10:    for all pairs of co-location vertices $(v_i, v_j)$ in $X_k$ do
11:        if $s(v_i) \cap s(v_j) \neq \emptyset$ then
12:            add distinct edge $e_{ij} = (v_i, v_j)$ to $G$
13:            if $s(v_i) \cup s(v_j)$ can run on the same device in $M$ then
14:                create new vertex $v_k$ from $v_i$ and $v_j$
15:                if $v_k$ exists in $G$ then
16:                    retrieve existing vertex $v_k$ from $G$
17:                else
18:                    add $v_k$ to set $X_{|s(v_k)|-1}$ and G
19:                end if
20:                add $v_i$ and $v_j$ to generator set $g(v_k)$
21:            end if
22:        end if
23:    end for
24:    find the smallest $i > k$ where $X_i \neq \emptyset$
25:    if no such $i$ exists, stop, else set $k = i$
26: end while
```

The algorithm checks the feasibility to deploy all service components of the union of $s(v_i)$ and $s(v_j)$ on a single IoT device. If it's feasible to select two co-location vertices at the same time, it creates vertex v_k as a new option to select all co-location options in its generator set at the same time. v_i and v_j are added to the generator set of v_k in order to keep track of how v_k is being created. The newly created vertex v_k is pushed to G and its corresponding layer according to its size of service components. Before the nested loop starts, for each vertex in each layer, it will create edges between it and all neighboring vertices of each generator vertex. In this way, the algorithm builds up a complex relationship of vertices between different layers. Finally, the algorithm finishes the transformation when all layers are settled, leaving the graph G as the co-location graph.

3.4.2 Selection Strategies

In our earlier study [44], we use a simple algorithm that treats every problem as a 2 co-locatable problem, which means every time the algorithm only selects an edge to co-locate. In this work, we take all possible co-location combinations into consideration, and use the solution strategies for the MWIS problem in our selection framework. In [82], researchers have studied three type of strategies and given their corresponding lower bounds. We briefly review them below.

1. GWMAX: the strategy selects each v_i that minimizes the function $W(v_i)/d_{G_i}(v_i)(d_{G_i}(v_i)+1)$. Once a node v_i is selected, it and its corresponding edges will be eliminated. When there is no edge left in G, the remaining nodes will form a maximum independent set.
2. GWMIN: this strategy selects each v_i that maximizes the function $W(v_i)/(d_{G_i}(v_i)+1)$. A node v_i will be selected in every iteration, and it will then be eliminated with its neighbors. The selected nodes during this process will return an independent set.

3. GWMIN2 is an extension of GWMIN by using different vertex-selecting rule. It selects each v_i that maximizes the function $W(v_i)/\sum_{w\in N^+_{G_i}(v_i)} W(w)$.

In all strategies described above, v_i represents the i_{th} node chosen from G. G_i is the G after $i-1$ round of node selection and update. The function $d_{G_i}(v_i)$ determine the degree of v_i in G_i. In GWMIN2, $N_{G_i}(v_i)$ denotes the neighborhood of v_i, and $N^+_{G_i}(v_i)$, $v_i \cup N_{G_i}(v_i)$.

3.4.3 Co-Location Selection Framework

We now present the general selection algorithms. It takes an FBP as input, splits the FBP into several subgraphs, in which every edge is co-locatable, and then builds corresponding co-location graphs. After that, for each graph, it uses the co-location selection strategy to select the maximum weighted independent co-location node set. Then, it selects a device to host all WuClasses in every co-location node in the maximum weighted independent set.

Since the graph construction algorithm has checked the feasibility of creating a vertex v for a co-location graph and the order of selecting co-location nodes to deploy will not impact the total energy saving, we just randomly pick a node to deploy at each round. (However, the order of selecting nodes would affect the maximum energy consumption among all devices; the problem will be investigated in our future work.) On Line 6 of Alg. 3.2, we could use any selection strategy that is good for a particular FBP structure or system setting. The flexibility provided by the framework allows a system to dynamically replace selection strategy at run time, which is desirable during the reconfiguration of an intelligent IoT system.

Algorithm 3.2 Selection Framework

Input: FBP $G(C, L)$ and device system M

Output: A pairing list P of C_i and its deployed device D_k

```
1:  $P = \varnothing$
2:  split $G$ into a list of sub-graphs $H$
3:  add all devices $D$ of $S$ to queue $Q_d$ in descending order of current energy cost
4:  generate co-location graph $G$' for every sub-graphs in $H$ and add them to list $L(G)$
5:  for all co-location graph $G_i \in$ L(G) do
6:      select MWIS $I_i$ from $G_i$ with a specific strategy
7:      for all vertex $v \in I_i$ do
8:          if $D = \{D_k | D_k$ can host every $C_i \in s(v)\} \neq \emptyset$ then
9:              select $D_k \in D$ that has the smallest energy
10:             for all $C_j \in s(v)$ do
11:                 add pair ($C_j$, $D_k$) to $P$
12:             end for
13:             update the current energy cost on $D_k$
14:         end if
15:     end for
16: end for
17: for all component $C_j$ without a deployment target do
18:     if $D = \{D_k | D_k$ can host $C_j\}$ then
19:         select $D_k \in D$ that has the smallest energy
20:         add pair ($C_j$, $D_k$) to $P$
21:     end if
22: end for
```

Figure 3.4 shows two mapping decisions P_1 and P_2. P_1 is selected by the the MWL algorithm [44]. It merges links L_{67}, L_{45} and L_{12} in order, and totally saves $30 + 20 + 10 = 60$ units of energy cost. P_2 is selected by GWMIN2. To reproduce the selection scenarios of GWMIN2, we use the function $f_k(v_i)$ to represent the current value of $W(v_i)/\sum_{w\in N^+_{G_i}(v_i)} W(w)$ for node v_i in the updated co-location graph after selecting k - 1 nodes. For the co-location graph in Figure 3.4, it is easy to see that v_6 has maximum value $f_1(v_6) = 30/\ (30 + 20) = 0.6$. After that, the node v_5 is removed due to the independence constraints. Then, the function value of v_4 is updated. After comparing the value all five nodes $f_2(v_1) = 10/(10 + 15 + 33) = 0.172$, $f_2(v_2) = 15/(10 + 15 + 33 + 18 + 20) = 0.156$, $f_2(v_3) = 18/(15 + 18 + 33 + 20) = 0.209$, $f_2(v_3) = 20/(20 + 33 + 15 + 18) = 0.232$, and $f_2(v_7) = 33/(33 + 10 + 15 + 18 + 20) = 0.343$, GWMIN2 strategy will find the optimal co-location decision which saves $30 + 33 = 63$ units of energy cost.

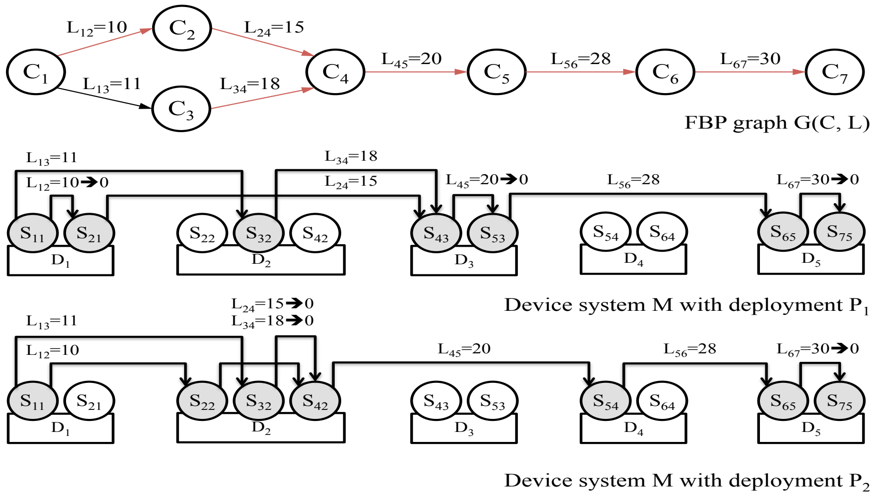


Figure 3.4: Co-location Solution Comparison

3.4.4 Mapping Remaining Services

Since not all components in a FBP may be selected for co-location, we need to select a device for those components that have not been mapped in the co-location decisions. On lines 17 - 22, the algorithm selects a device with the lowest current energy load for each single component C_i by using the same selection strategy for co-location nodes. In this way, we could achieve a better energy balance on all devices.

For the example of Fig. 3.3, the mapping decision is to co-locate (C_2, C_3, C_4) on devices D_2 and (C_6, C_7) on device D_5. After that, we still need to select devices for mapping components C_1 and C_5. In this case, since D_1 and D_4 don't have any load, we simply select them for C_1 and C_5 respectively.

3.5 Simulation Study

We have implemented the sensor selection framework in Algorithm 3.2, and used selection strategies including MWL, GWMAX, GWMIN and GWMIN2. As an extended study of our previous work [47], we compare six mapping algorithms shown in Figure 3.5. The maximum weight scoring function algorithm (MAXIMUM) is to select the maximum weighted link from the co-location graph, and the one layer maximum weight scoring (ONE LAYER) is to select the maximum weighted link from the one layer co-location graph that only co-locates two neighboring nodes. In this section, we show how we set up the simulation environment, the consideration for determining system parameters and performance metrics, and present the performance comparison for all six algorithms.

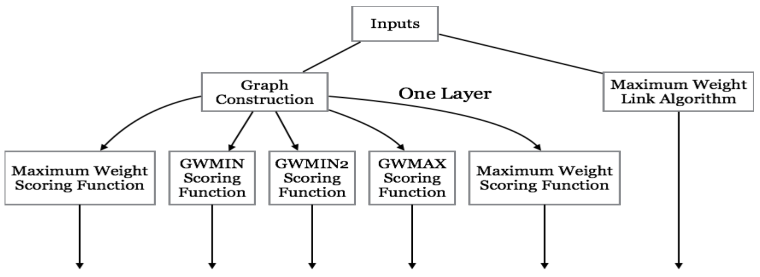


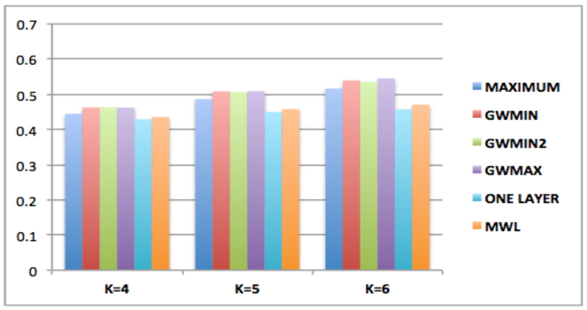
Figure 3.5: Mapping Algorithms

3.5.1 Simulation Setup

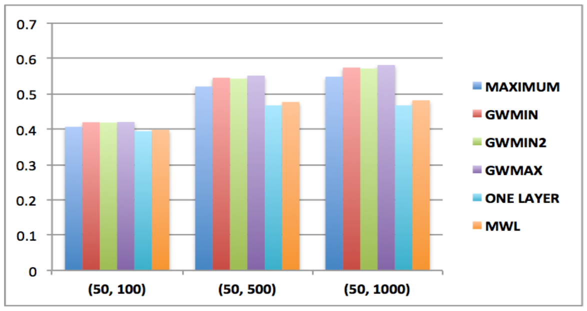
We generate a simulation system with n components and m devices as follows. On each device D_j, we randomly select K different WuObjects as available services on it. K is the upper bound of co-locating size and is viewed as memory constraint for each device. We then use JGraphT to generate different types of flow graphs, including 1000 instances of linear, star or random structures as FBPs with the size half of the total WuClasses in the system. Linear FBPs are common for data transmission applications. Star FBPs are often used for system and environment monitoring application. We also use random FBPs as the topology for general intelligent applications.

In a WuKong system supporting Z-wave communication, a normal information exchange message is about 10 bytes. Including the header of Z-wave protocol, the total size of Z-wave packet is about 40 bytes. For system management messages, the payload is bigger but can't exceed the maximum size of Z-wave payload which is 64 bytes. Therefore, the size of a WuKong message is about 40 - 100 bytes in normal cases. Before deployment, we assume application developers are responsible for finding the data rate of each component. In the simulation, we assume data rate of each component is one message per second. Based on these considerations, we use uniform distribution $d_1 = \mathrm{U}(40, 100)$ to generate the data volume of each link L_{ij} in an FBP. Even though there are only about 6 - 7 types of messages with different sizes, the normal distribution would randomize the filter rate for components with a threshold. After that, we calculate the corresponding transmitting cost t_{ij} and receiving cost r_{ij} using Eq. 3.1 and Eq. 3.2.

Wukong ecosystem aims to provide flexible application deployment and runtime management for large scale IoTs which involves hundreds or even thousands of devices connected. In this paper, the data for 50 components in FBP and systems scales from 100 to 1000 physical devices is reported. On each device, there are a set of K WuClasses with K from 4 to 6.



(a) T Ratio Comparison for random FBP $(n, m) = (50, 100)$



(b) T Ratio Comparison for random FBP $K = 4$

Figure 3.6: T-Ratios for different selection algorithms.

3.5.2 Performance Metrics

In this paper, we compare the total saved energy cost ratio (T-Ratio) and the largest energy cost ratio (L-Ratio) defined as follows:

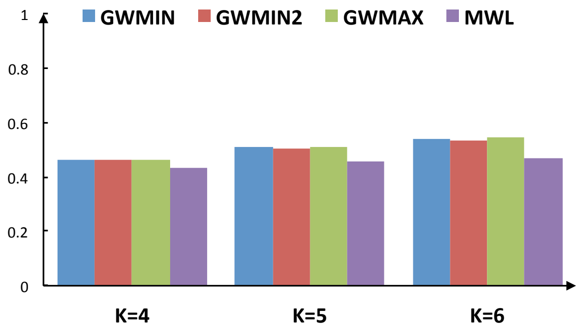
1. **Total Saved Energy Cost Ratio** (T-Ratio): is the percentage of saved energy cost of the whole FBP with service co-location compared to the FBP original cost without any co-location.

2. **Largest Energy Cost Ratio** (L-Ratio): is the percentage of energy cost in the device with the highest energy cost by applying different algorithms.

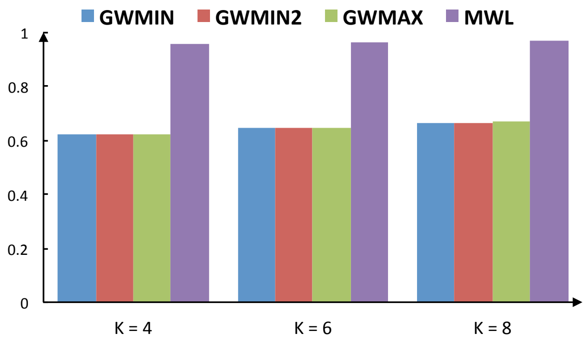
Using T-ratios and L-ratios, the higher a ratio value is, the more energy is saved. The energy saving reduces as the probability of service co-location reduces.

3.5.3 Performance Comparison

Figure 3.6 shows T-ratios in ramdom structure of FBP. It can be seen that the three methods GWMIN, GWMIN2 and GWMAX perform better than the MWL algorithm and its variants MAXIMUM and ONE LAYER. When K is small, the difference of performance between the three methods and MWL algorithm is not so obvious. As K grows to 6, we find that the overall energy saving performance is growing since the likelihood of co-locating multiple WuClasses on a device increases. Moreover, as K grows, the difference of performance between the three methods and MWL algorithms becomes more obvious. The reason is that the MWL greedy algorithm considers the selection as a 2-co-locating problem. It is natural to expect that energy saving depends on the likelihood of service co-location in a system. If an application is a small FBP, or the system has a large number of unique sensors, the chance for sensor co-location, and thus energy saving, is small.



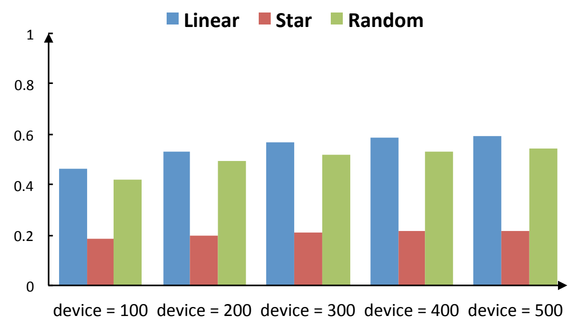
(a) T-Ratio



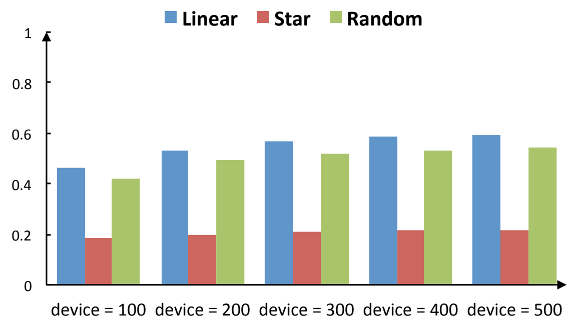
(b) L-Ratio

Figure 3.7: Performance of different K sizes for linear FBP's.

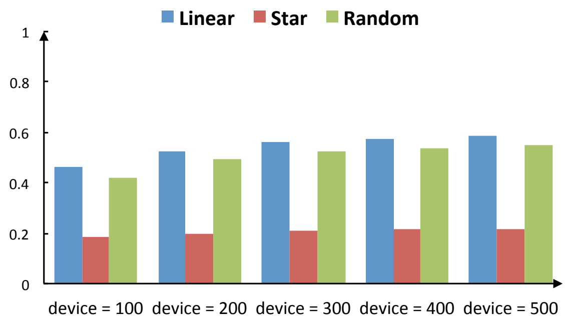
If we compare the result of using the same selection strategy of maximum weight link on two different co-location graphs (fully built co-location graph and one layer co-location graph), we may discover that a fully built co-location graph helps us find better solutions. It is because such a fully built graph includes co-location decision that co-locates multiple services. From Figure 3.6, we can see that GWMIN, GWMIN2 and GWMAX algorithms always surpass MAXIMUM, MWL, and ONE LAYER. Therefore, we focus our performance study on GWMIN, GWMIN2, GWMAX and MWL in Figure 3.7, which shows the T-ratio and L-ratios on linear structure of FBP's. A similar performance pattern for T-ratio can be seen.



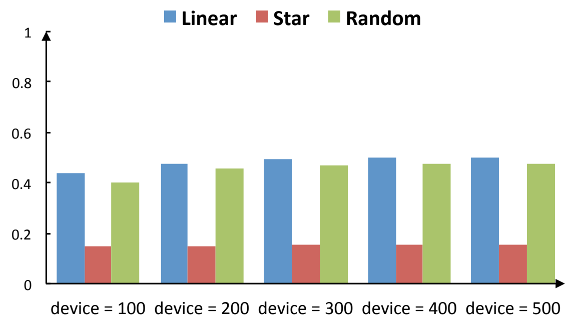
(a) GWMIN algorithm



(b) GWMIN2 algorithm



(c) GMAX algorithm



(d) MWL greedy algorithm

Figure 3.8: T-Ratios for different FBP structures.

We have also studied how the FBP structure affects T-Ratios. We compare linear, star and random FBP structures in Figure 3.8. We see that the three new algorithms and MWL greedy algorithm perform worse in the star structure FBPs when the number of device grows from 100 to 500. The intuition for this fact is that there can be only one choice for the central component in star FBP to co-locate with. That means there only exists one co-location in star shape FBP. Moreover, MWL has a relatively bad performance in all shapes of FBP structure for all numbers of devices. This is because the MWL algorithm can only consider a single edge of FBP in each round.

(n, m)	GWMIN	GWMIN2	GWMAX	MWL
(50, 100)	3.182ms	3.223ms	3.085ms	0.07ms
(50, 200)	9.126ms	9.172ms	8.852ms	0.104ms
(50, 300)	17.454ms	18.554ms	15.855ms	0.154ms
(50, 400)	32.254ms	27.347ms	22.643ms	0.136ms
(50, 500)	49.856ms	34.809ms	31.794ms	0.185ms
(50, 1000)	192.554ms	127.458ms	123.264ms	0.346ms

Table 3.1: Scalability with K = 4 and different (n, m)

K	GWMIN	GWMIN2	GWMAX	MWL
4	21.73ms	18.525ms	13.923ms	0.303ms
5	31.631ms	24.571ms	18.776ms	0.213ms
6	44.669ms	32.212ms	28.955ms	0.196ms

Table 3.2: Scalability with (n, m) = (50, 100) and different K

Beside the metrics for energy saving, we have also studied the execution times for different algorithms. It is an important factor if we want to deploy an application with many services. In Table 3.1, we show the performance of algorithms in six size settings. Each row of (n, m) shows the system with n components and m devices. For each case, we show the average execution time for each algorithm. We can see that the execution time grows linearly for the first three cases. But it grows to more than 192 milliseconds in the last case, which is because the combination of selections grows exponentially. In Table 3.2, we study how K affects the execution time. The greedy algorithms take longer to consider more co-location decisions as K increases while the MWL uses about the same time.

Protocol	Input Power (dbm)	Output Power (dbm)
Zwave	-22.0dBm to -2.0dBm	10μW to 612μW
XBee	0dBm	1mW
Xbee Pro	18dBm	63mW
Low Power Wifi	10dBM to 0dBm	501μW to 10mW

Table 3.3: RF Energy Consumption per Bit

In summary, from the performance of energy saving and reasonably short computation time, we believe the co-location consideration is good for many IoT systems that need to support run-time application mapping, deployment and reconfiguration in a smart IoT environment.

3.6 Summary of Energy Mapping

This section presents an energy sentient methodology for selecting and deploying flow-based IoT applications on sensor devices. Since energy is one of the most important resources for running IoT devices, we propose a mapping strategy that tries to minimize the total energy cost for communication by co-locating neighboring services on the same node. We have modeled the co-location problem on multi-hop networks as a quadratic programming problem, so that it can be solved by integer programming. For single-hop networks, we identify co-locatable components of an FBP to construct a co-location graph, and develop the selection framework using efficient MWIS algorithms to decide service co-locations. Our simulation study shows that the MWIS algorithms can save 10% more communication energy than our previous solution.

Chapter 4

Mapping in Scalable Reconfiguration

One of the ways to proactively reconfigure IoT systems is through reprogramming. Researchers have proposed various algorithms [93, 86] to improve the efficiency of reprogramming in the Wireless Sensor Network (WSN). In IoT systems, the speed of reprogramming to a device is limited by low bandwidth of last-hop IoT networks, such as Zwave and Xbee, in which gateways are used as a relay to sequentially reprogram each target device. Carefully choosing the set of devices to reprogram may relieve the overhead of gateways, thus reducing the reprogramming latency.

Consider an earth quack protection application that uses gyroscope to detect emergent situations. Four components are used in the application: component C_1 checks for shaking by recording the deviation of three axes in gyroscope, component C_2 accumulates sufficient evidence within a period of time, component C_3 is a threshold, and C_4 responses to signals from C_3 by triggering emergent controller. In Figure 4.1, region A and B need to be reprogrammed. Region A contains devices $\{D_1, D_2, D_4\}$ connected by gateways 1 and 2. Region B contains devices $\{D_3, D_5, D_6\}$ connected by gateways 2 and 3. If we map the application to $(S_{11}, S_{22}, S_{35}, S_{46})$ in region A and $(S_{13}, S_{24}, S_{37}, S_{48})$ in region B, each region has a balanced mapping, in which only two devices need to be reprogrammed under each gateway from a local view. But in a global view, gateway 2 actually need to sequentially reprogram all four devices. After observation, we may easily find a global optimal solution, which is to map the application to $(S_{11}, S_{22}, S_{32}, S_{46})$ in region A and $(S_{17}, S_{24}, S_{37}, S_{48})$ in region B. In this case, we only need to reprogram two devices under each gateway.

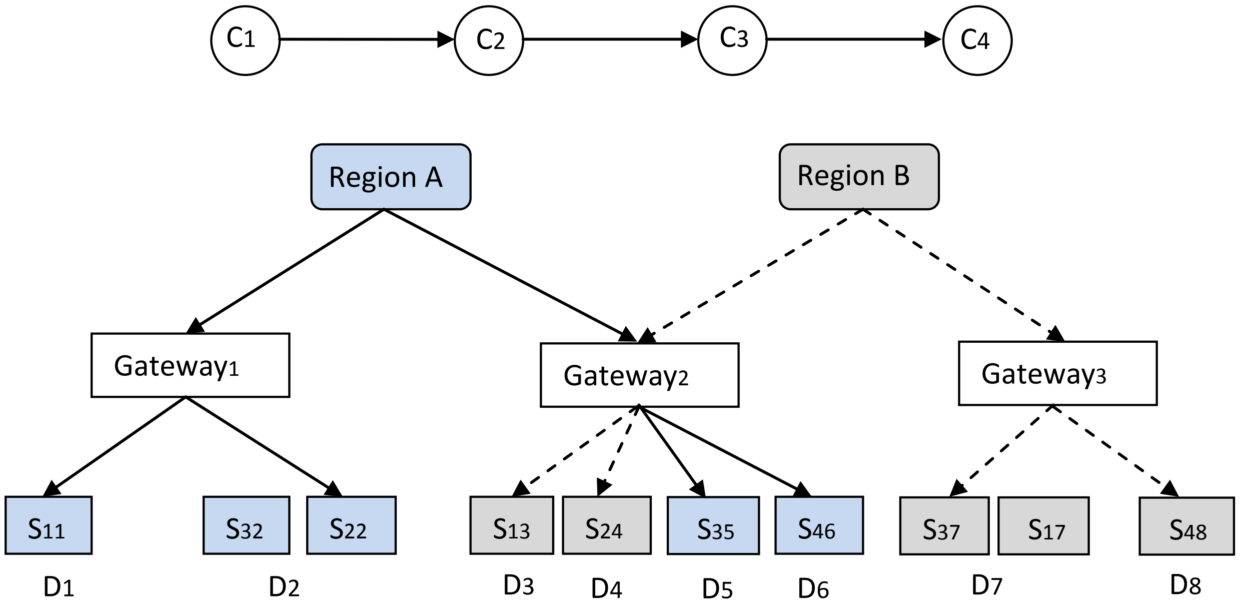


Figure 4.1: Reprogramming Latency Aware Mapping Example

We define the problem of minimizing the maximum number of reprogram devices under each gateway as problem P_1, and model it as an Integer Linear Programming (ILP) problem. Given a building with 200 rooms (regions), assume there are a total of 10 gateways and 10 devices per region, our algorithm only needs to reprogram 60 devices compared to 80 and 100 by two other greedy algorithms.

In the mean time, each deployed application should meet its end to end latency constraint to provide responsive and secure service for people. In this study, we further model the run time deadline constrained low latency reprogramming problem P_2 as an Integer Quadratically Constrained Programming (IQCP) problem. However, there is no such thing as a free lunch, for the computation time of mapping should also be included in the reprogramming latency. For a big region (problem space), resolving the IQCP problem takes even more time than the latency of propagating code to each device. Thus, we devise two heuristic algorithms to reduce the time complexity of mapping algorithm through linear relaxation and service relocation. To the best of our knowledge, this is the first work to design a remote programming method considering both the service availability and the need of guarantee run-time latency.

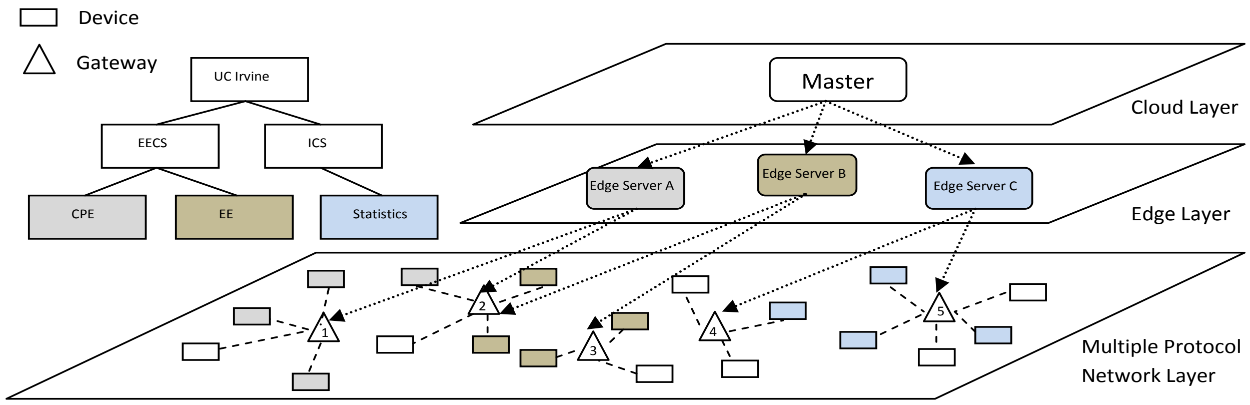


Figure 4.2: System Model

4.1 Deployment Cost Model

In this section, we first introduce application model, network architecture, then reprogramming latency formulation.

4.1.1 Network Architecture

In our study, IoT systems in Smart Cities are modeled as distributed systems consisting of a set of sensor/computing/actuator devices that are placed on different locations in target regions, connected by RF communication channels. The whole system is composed of servers and devices scattered in three sub-layers:

- Cloud Layer. System Master and its supporting services are hosted in the Cloud Layer. Master contains global information of Smart Cities. For the organizational interoperability and management consideration, all of the organizations are organized as a hierarchical region tree. As shown in Figure 4.2, UC Irvine contains two sub regions, which are Electrical Engineering and Computer Science (EECS) department and Information and Computer Science (ICS) department. In the leaf layer of the region tree, there are Computer Engineering EE, Electrical Engineering (EE), and Statistic major as regions. Each leaf region contains a set of devices, and each upper layer region has the management authority on devices of its sub regions. Once an emergent event is identified, the master will issue a request of mapping and reprogramming to target regions.

- Edge Layer. Edge servers are arranged in this layer to enable data analytic and knowledge generation to occur near physical devices. During remote programming, edge servers are responsible for parallel performing mapping algorithms for each target region. After mapping, a edge server will start remote programming to each selected device through talking with gateways.

- Multiple Protocol Transmission Network Layer. Multiple Protocol Transmission Network (MPTN) Layer contains physical devices and gateways. In MPTN, devices paired with different protocol are inter-connected by gateways. In Figure 4.2, devices that belong to a particular region are covered by more than one gateway. For example, devices in region CPE are connected with gateways 1 and 2. Thus, edge server A need to connect with both gateways 1 and 2 for remote programming.

An IoT system has a set of physical devices D_k. On each device, there may be several services available for sensing or computing. Service S_{ik} belongs to WuClass Y_i and is hosted on device D_k. As shown in Figure 4.1, a physical device D_k may host multiple sensors or computing services, called *WuObjects*, that can be used for fulfilling some WuClasses. For example, S_{22}

and S_{32} on device D_2 can be used for C_2 and C_3 components, respectively. WuClass Y_i can be classified into two categories: virtual and physical. On one hand, a component C_i belongs a physical WuClass C_i can only be mapped to devices D_k that physically deploy service S_{ik}. On the other hand, a virtual component can be mapped to any reachable device of a region.

The proposed work in this chapter is to derive the sensor mapping strategy in the edge given the information of services on devices of a set of regions. The sensor mapping strategy aims to minimize the reprogramming latency, while satisfying the service requirement of emergent application in each target region.

4.1.2 Reprogramming Latency Formulation

As shown in Figure 4.2, the latency of reprogramming a target region R_m contains three parts: 1) Time T_{m1} of sending mapping request from master to edge server; 2) Time T_{m2} of sensor mapping and code generation in an edge server; 3) T_{m3} is the time that an edge server reprograms each selected device through gateways.

If we assume T_{m1} is ignoble, the reprogramming latency from master to region R_m can be expressed as:

$$T(R_m) = T_{m2} + T_{m3} \tag{4.1}$$

To have an exact estimation of reprogramming latency, we profile T_{m2}, T_{m3} in Zwave network of WuKong system. In the profiling setting, all devices are connected through Zwave gateways. These devices are arduino compatible and run with Darjeeling java virtual machine. Reprogramming is achieved by writing a Darjeeling Archive File (DJA) as an application into the EEPROM of each of device, then every device reboots and reloads the new application into Darjeeling.

Given an application (FBP) as a mapping request, the DJA file contains the meta-data of links of FBP, initial value of properties, mapping result for each component, and binary code of each type of virtual Wuclass. No matter the mapping result, the code generation time $T_{gen}(R_m)$ and the size of DJA file that need to be sent to device are fixed. Therefore, the part of latency that is optimizable only contains two parts: the mapping time $T_{map}(R_m)$ and T_{m3}.

Assume the latency of sending one package from a gateway D_k to a device is t and the size of package is p, the time of reprogramming a device D_k with a DJA file is:

$$T(D_k) = Size(DJA)/p * t \tag{4.2}$$

In WuKong system, the MPTN package size p is 36 bytes and the zwave transmission latency is 110ms. In Figure 4.1, we can see that the linear FBP of (C_1, C_2, C_3, C_4) need be mapped and reprogrammed to both region A and region B. Region A contains devices D_1, D_2, D_4. If a mapping solution maps the FBP onto a service combination of (S_{11}, S_{22}, S_{34}, S_{44}) in region A, then the DJA file needs to firstly be forwarded to both gateway 1 and gateway 2. After that, each gateway will sequentially reprogram each target device connected by itself. In this case, gateway 1 needs to reprogram D_1, D_2 and gateway 2 need to reprogram device D_3. Assume the set $S(G_n, R_m)$ is the set of devices that are connected with gateway G_n and selected for mapping results in region R_m, the reprogramming time of gateway G_n for the region R_m is:

$$T_{Gn}(R_m) = \sum_k (T(D_k)) \forall D_k \in S(G_n, R_m) \tag{4.3}$$

Assume the set $S_G(R_m)$ contains all gateways that need to be used for package forwarding of a reprogramming request in region R_m. Since the package forwarding concurrently happens in each gateway, then the T_{m3} can be expressed as:

$$T_{m3} = \max_n (T_{Gn}(R_m)) \forall G_n \in S_G(R_m) \tag{4.4}$$

From the equation, we may derive that we need to minimize the maximum number of reprogram devices under each target gateway. This idea inspires our work to find the optimal sensor mapping strategy to minimize the overall programming latency.

4.2 Sensor Mapping Without Considering Run-time Cost

In this section, we firstly introduce the concept of congestion zone, which is the minimum sensor mapping problem, then present the problem definition of problem P_1 and ILP formulation, and analyze the problem complexity.

4.2.1 Congestion Zone

Given a target mapping request, a gateway is congested if and only if it is shared by more than one region and each region contains target devices connected by the gateway. A congestion zone is a set of regions in which any region shares at least one congestion gateway with one of the other regions in the zone. Beside this, any two congestion zones do not share any congestion gateway.

In Figure 4.1, gateway 2 is congested for both region A and B having target devices under it. Thus, region A and region B form a congestion zone. The reason behind defining the congestion zone is that an optimal mapping strategy that applies to regions separately can't guarantee global optimality. Thus, a congestion zone is the minimum scope in which we should apply sensor mapping strategy.

4.2.2 Problem Definition

Given a congestion zone Z, assume it has N gateways $G_n(1 \leq m \leq N)$ and M regions $R_m(1 \leq n \leq M)$. Each device $D_k(1 \leq k \leq K)$ connects one of gateways, and belong to one of the regions. The problem studied in this chapter is to determine a mapping solution so that there are a sufficient number of services in each region in Z. Let x_{ik} be a binary variable, which $x_{ik} = 1$ indicates that service S_{ik} is selected on device D_k. Let y_k be another binary variable, which $y_k = 1$ indicates that device D_k needs to be reprogrammed. Then, we need to guarantee that each component in the FBP has been mapped to a service in each region R_m.

Under a mapping solution $\{x_{ik}\}$, the device D_k need to be reprogrammed if and only if at least one of the service x_{ik} is selected. In this case, each gateway G_n has a reprogramming load

$$\alpha_n = \sum_k (y_k) \tag{4.5}$$

i.e., the number of reprogram devices under it, which indicates the reprogramming latency for gateway G_n. Essentially, we hope to reduce α_n to save reprogramming latency. Thus, our objective is to minimize the maximum load $\max(\alpha_n)$, for the purpose of load balancing on each gateway in a congestion zone Z.

4.2.3 Computation Complexity

Before presenting our algorithms for the problem, we first show that the problem is computationally intractable. Given a congestion zone Z, assume there are M regions in it.

Theorem 2. *The sensor mapping problem P_1 is NP-hard in the strong sense even $M = 1$.*

Proof. We prove the theorem by a reduction from the Not-All-Equal 3SAT (NAE3SAT) problem, which is known as NP-complete in the strong sense.

NAE3SAT: For a given set U $= u_1, u_2, u_3 \ldots u_n$ of n binary variables, and a collection of clauses $C_1, C_2, C_3 \ldots C_m$ defined over literals $u_1, u_2, u_3 \ldots u_n, \bar{u_1}, \bar{u_2} \ldots \bar{u_n}$ such that each clause has three literals $C_i = u_{i1} \vee u_{i2} \vee u_{i3}$, the question is whether there exists a true assignment for $C = C_1 \wedge C_2 \wedge C_3 \ldots \wedge C_m$ such that each clause C_i has at least one true literal and at least one false literal.

Given an instance of NAE3SAT, we can construct an instance of the decision version of problem P_1 as follows: Let there be 2n Device each corresponding to a literal. With a bit of abuse of notation, we also denote the Device as $u_1, u_2, u_3 \ldots u_n, \bar{u_1}, \bar{u_2} \ldots \bar{u_n}$, all of them are in the same region. There are n gateways, each gateway G_i connects two devices $u_i, \bar{u_i}$. Let there be n + m components in a FBP.

1. Each component S_i, i = 1, . . . , n, can be provided by device $u_i, \bar{u_i}$
2. Each component S_{n+i}, i = 1, . . . , m, can be provided by device u_{i1}, u_{i2}, u_{i3}, where C_i $= u_{i1} \wedge u_{i2} \wedge u_{i3}$. Let each component need on instance at any time.

The question is whether there is a mapping within the region and $max(\alpha_i) \leq 1$.

1. Suppose that there is a true assignment for NAE3SAT. Then we can have a mapping with R = 1. In the assignment, if $u_i = 1$ (imply $\bar{u_i} = 0$), means device ui need reprogramming. If $\bar{u_i} = 1$ (imply ui = 0), means device i need reprogramming. In such a mapping, for any service s_i, i = 1, . . . , n, map to device u_i if u_i is true, otherwise map to i. For any service s_{n+i}, i = 1, . . . , m, because one of literals u_{i1}, u_{i2}, u_{i3} is true, At least one of Device u_{i1}, u_{i2}, u_{i3} is already be selected for hosting service, then we simply map s_{n+i} that particular device. Therefore, the mapping is feasible. Because each gateway G_i needs to reprogram only one of $u_i, \bar{u_i}$, we have $G_i = 1$ for all gateways.
2. Suppose that there is a mapping R = 1 and $max_(\alpha_i) \leq 1$. So any gateway only need to reprogram one device. In the corresponding NAE3SAT problem, we can let $u_i =$ 1, if device u_i need to reprogram, $\bar{u_i} = 0$, if device $\bar{u_i}$ need to reprogram. The above analysis shows that this is a feasible solution to NAT3SAT.

□

4.2.4 Integer Programming Formulation

Given a congestion zone in an IoT system, its internal network topology should be fixed. We use vector $A_n = \{a_{n1}, a_{n2}, \dots, a_{nk}\}$ to denote the connection status of gateway G_n to each device D_k. If $a_{nk} = 1$, then device D_k connects to gateway G_n. We use vector $B_m = \{b_{m1}, b_{m2}, \dots, b_{mk}\}$ to denote the relationship between region R_m and each device D_k. If $b_{mk} = 1$, then region R_m contains device D_k.

For the set of devices in the congestion zone we are considering, the services on each device should be also predefined. We use vector $C_i = \{c_{i1}, c_{i2}, \dots, c_{ik}\}$ to denote how instances of service S_i are installed. If $c_{ik} = 1$, it means one instance of S_i is installed on device D_k.

For each mapping request, a FBP contains a set of component C_i. We use a Vector Q = $\{q_1, q_2, q_3, \dots, q_i\}$ to represent the resource requirement for each region.

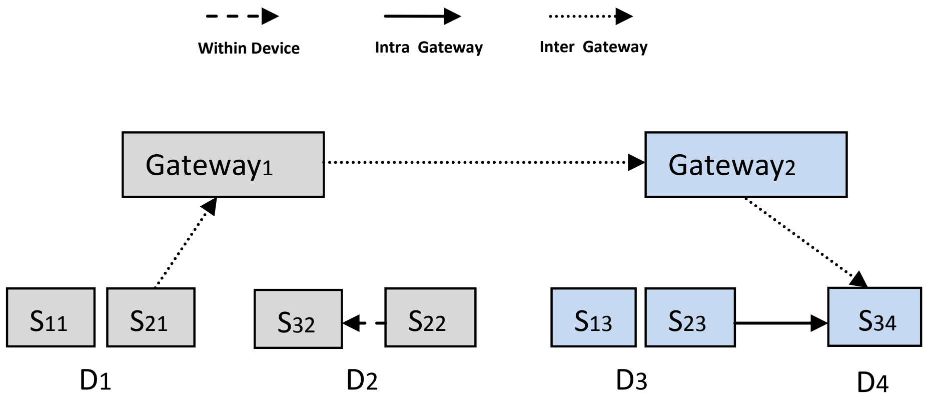


Figure 4.3: Run-time Latency Example

In ILP, the selection variable x_{ik} is defined as:

$$x_{ik} = \begin{cases} 1, \text{if service } S_i \text{ on device } D_k \text{ is selected.} \\ 0, \text{otherwise.} \end{cases} \tag{4.6}$$

we use another binary variable y_k

$$y_k = \max\{x_{ik}\}, \forall 1 \leq i \leq I \tag{4.7}$$

to determine whether device D_k needs to be reprogrammed. Eq.4.7 guarantee a device D_k is counted as long as one of the services on it is selected.

The sensor selection problem is formulated as follows:

$$\min{(\alpha)} \tag{4.8}$$

subject to:

$$B_m \cdot X^T = Q, \forall 1 \leq m \leq M \tag{4.9}$$

$$\alpha_n = \sum_{D_k \in R_n} (y_k) < \alpha, \forall 1 \leq n \leq N \tag{4.10}$$

$$x_{ik} = \{0, 1\}, \forall 1 \leq i \leq I, 1 \leq k \leq K \tag{4.11}$$

In the formulation, Eq. 4.9 ensures that we meet the resource requirement of an application within each region, and Eq. 4.10 evaluates the maximum number of reprogramming devices among all gateways.

4.3 Sensor Mapping Considering Run-time Cost

In emergency management applications, the end to end latency is the most significant quality of service (QoS) for building responsive action to emergent event. In this section, we will discuss how to model the run-time latency and apply the constraint in mapping model, and heuristic algorithms that reduce computation complexity.

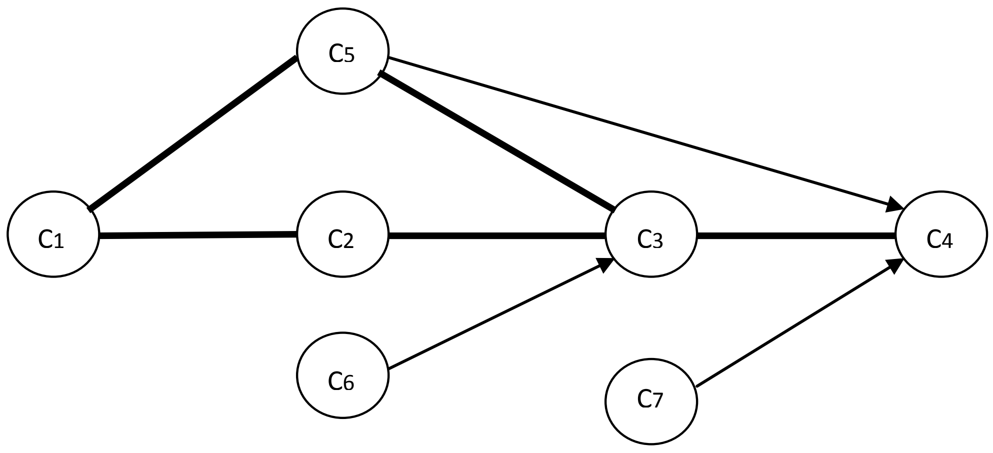


Figure 4.4: Dominante Path Example

4.3.1 Run-time Latency Model and Constraint

In smart home, smart building, and factory automation, Z-Wave and ZigBee are the two of most popular low power wireless communication technologies. They have different transmission ranges and data rates. We profile the Z-Wave transmission latency between two devices as 110ms. Given a FBP, its end to end latency depends on the transmission time of each link L_{ij}, which is further determined by how it is mapped to physical services on devices. In our study, we denote the latency coefficient $\beta(L_{ij})$ as

$$\beta(L_{ij}) = \begin{cases} 0, \text{if } C_i,\ C_j \text{ on the same device.} \\ 1, \text{if } C_i,\ C_j \text{ in one sub-network.} \\ 2, \text{if } C_i,\ C_j \text{ in two sub-networks} \end{cases} \tag{4.12}$$

As shown in Figure 4.3, a region contains four devices $\{D_1, D_2, D_3, D_4\}$ covered by two sub-networks. Assume there is a link L_{23} in an emergent application. It can be mapped onto three places $\{(D_2), (D_3, D_4), (D_1, D_4)\}$. If both of C_2 and C_3 are mapped to device D_2, the latency is ignoble. Thus, latency coefficient $\beta(L_{ij})$ is 0 in this case. Otherwise, $\beta(L_{ij})$ denotes the number of hops of the transmission link for practical latency is linear to it.

Given an FBP as a DAG, it can be decomposed to be a set of paths P. A path P_u is a linearly connected links. For example, the path (C_1, C_2, C_3, C_4) contains a set of sequentially connected links $\{L_{12}, L_{23}, L_{34}\}$ in Figure 4.4. We denote transmission latency of path P_u as

$$\beta(P_u) = \sum_{L_{ij} \in P_u} \beta(L_{ij}) \tag{4.13}$$

Assume the end to end deadline of a FBP is specified as 500 milliseconds, then we may estimate the maximum number of hops of the longest path is $500/110 = 5$ hops in the Z-wave network. In this way, we may find all of the dominant path P_D that might miss deadline and use contraints to prevent it happen in mapping result. Given the maximum hop H as the upper bound, every path whose length is larger than H/2 is a dominant path. For example, there are two dominant paths (C_1, C_2, C_3, C_4) and (C_1, C_5, C_3, C_4) in Figure 4.4, for their length are 3, which is larger than $5/2 = 2.5$.

As we discussed, the second problem P_2 is to find minimum reprogram latency mapping solution with the consideration of both service availability and End to End run-time latency. It can be resolved by adding communication latency of each dominate path in original problem P_1. Given a congestion zone, we define O_{ij} as a set of device pair $P_{nm}(D_n, D_m)$ within a gateway that may host link L_{ij}, and define R_{ij} as a set of device pair $P_{nm}(D_n, D_m)$ cross gateways that may host link L_{ij}. In the physical network setting, the Eq. 4.12 can be rewritten as

$$\beta(L_{ij}) = \sum_{P_{nm} \in O_{ij}} x_{in} * x_{jm} + 2 * \sum_{P_{nm} \in R_{ij}} x_{in} * x_{jm} \tag{4.14}$$

Thus, for each path P_u within P_D, we need to add one contraint as

$$\beta(P_u) = \sum_{L_{ij} \in P_u} \beta(L_{ij}) \leq H \tag{4.15}$$

The Eq. 4.14 means P_2 becomes a Integer Quadratically Constrained Programming (IQCP) problem. However, we could transform the problem to integer linear programming by rewriting the problem through introducing new variable y_{ikpm} and constraints to take the value $x_{ik}x_{pm}$ for every combination of x_{in} and x_{jm}, and y_{ikqn} for value $x_{ik}x_{qn}$ also. The equivalency of two formals have been proven in [18]. Then Eq. 4.14 can be replaced by

$$\beta(L_{ij}) = \sum_{P_{nm} \in O_{ij}} y_{injm} + 2 * \sum_{P_{nm} \in R_{ij}} y_{injm} \tag{4.16}$$

Besides formal constraints defined, we need to have more constraints for each y_{ikpm} and y_{ikqn} as below:

$$y_{ikpm} \geq 0 \tag{4.17}$$

$$x_{ik} - y_{ikpm} \geq 0 \tag{4.18}$$

$$x_{pm} - y_{ikpm} \geq 0 \tag{4.19}$$

$$1 - x_{ik} - x_{pm} + y_{ikpm} \geq 0 \tag{4.20}$$

These four equations ensure the value of y_{ikpm} to be exactly the value of x_{ik} * x_{pm} in any case. Once replace every x_{ik} * x_{pm} with its corresponding y_{ikpm}, the object function is actually ensured to be optimized under the same setting of $x_i k$ with original object function. Therefore we could replace x_{ik} * x_{pm} with y_{ikpm}.

4.3.2 Round-Up Algorithm Based on Linear Programming Relaxation

Because of the NP-hardness of the problem P_2, we design an approximation algorithm based on linear programming (LP) relaxation. The algorithm contains two stages. In the first stage, we resolve its LP relaxation by removing the integral constraints Eq. 4.11. It is replaced by

$$0 \leq x_{ik} \leq 1, \forall 1 \leq i \leq I, 1 \leq k \leq K \tag{4.21}$$

Algorithm 4.1 Round-Up Algorithm for problem P_2

```
1: Solve the LP relaxation Eq.(8)-(10), (15) and (16) to obtain a fractional solution $\{x_{ik}^L\}$
2: set $x_{ik}^H = 1$ for the d largest $x_{ik}^L$
3: set other $x_{ik}^H = 0$
```

In the second stage, a greedy round-up algorithm is designed to find a integral solution based on the linear solution from first stage. We use x_{ik}^L to represent the optimal solution to the LP relaxation, where some x_{ik}^L may not be integers, and use x_{ik}^H to represent the solution obtained based on rounding x_{ik}^L. In the round-up algorithm 4.1, we consider the service available of each component C_i. Given a zone with d regions, the first d service which can provide S_i and have the highest x_{ik}^L will be selected.

4.3.3 Heuristic Algorithm for Relocating Services

The Round-Up algorithm not only breaks the optimality of latency minimization, but also break the deadline constraints for dominant paths in Eq. 4.16. Given any $x_{ik}^H = 1$, it means we select the service S_{ik} on device D_k for component C_i in the region, to which D_k belong. If we choose another available device $D_{k'}$ in the same region to replace D_k for component

C_i, we may set $x^H_{ik'} = 1$ and reset x^H_{ik}. By relocating component C_i, it will not only keep the availability constraint in the region, but also will affect the run-time latency of some paths in the application. We thus aim to develop a heuristic algorithm that further improve the performance by relocating services among devices.

Algorithm 4.2 Service Relocation Algorithm

Input: A list of path P_D as the dominant paths of a FBP
Input: A set of variable x^H_{ik}
Input: A empty set of component fixed C_i as S_c
Output: A set of variable $x^H_{ik'}$ after relocation

```
 1: sort the list of path $P_i \in P_D$ in decent order of execution hops
 2: for all dominant path $P_i \in$ sorted $P_D$ do
 3:    if  $P_i$.length lg H then
 4:       for all adjacent components $C_i, C_{i+1}, C_{i+2}$ on path $P_i$ do
 5:          if All of $C_i, C_{i+1}, C_{i+2} \notin S_c$ then
 6:             Relocate $C_{i+1}$ to reduce run-time hops
 7:             Add three components into $S_c$
 8:          end if
 9:          i = i + 3
10:       end for
11:    end if
12: end for
```

After applying the result x^H_{ik} from round-up algorithm back to components of FBP, we know physical devices that host each component. With the known network architecture, we may calculate communication hops of each dominant path P_i of the FBP. In the heuristic algorithm 4.2, we firstly sort the dominant paths by the decent order of physical communication hops. For each path whose physical communication hops are larger than H, we perform relocate algorithm for each sub-path C_i, C_{i+1}, C_{i+2} that contains three unvisited adjacent components in path P_i.

Each time, we only consider relocating the middle component C_{i+1}. The basic idea is to co-locate C_{i+1} to the same device or gateway of C_i, C_{i+2}. When C_i, C_{i+2} are under two different gateways and both of them has available device to host C_{i+1}, we choose the the gateway with smaller load to reduce overall reprogramming time.

4.4 Simulation of Scalable Deployment

In this section, we introduce the effectiveness of our heuristic algorithms through simulation. We firstly introduce the design of simulation, and the algorithms we want to compare.

4.4.1 Simulation Setup

Our study is performed for systems with 100 congestion zones, in each of which there are 10 gateways. All the performance number are averaged on all of 100 zones. Under each gateway, there are 10 to 200 devices, which depends on the setting for each study group. At the mean time, we fix the region size as 10 devices, which means the number of regions grows linearly with the number of devices under each gateway. For each region, we need to deploy an application instance.

In the simulation, we consider three types of application environment: smart home, smart building, smart factory, in which the fault tolerant requirement grows. Given one type environment, we define two parameters service density θ and replica number λ to generate system settings, including sparse(2, 1), medium(3, 2) and dense(5, 3). For each setting, the first number means the minimum number of services on each device, and the second number means the minimum number of replica of each type of service in a region.

Since one congestion zone is an independent problem space, we generate a network architecture for it separately. Given a setting coefficient (θ, λ), we randomly choose θ number of services on a device, and make sure there are at least λ replica for each type of WuClass. Then, we generate a set of regions R_m according to the formula number(D_k) / 10, and each devices are randomly assigned to a region. To construct a real MPTN network, we need to determine how devices in region connect with gateways. In our study, we do consider the network in a physical flat, in which devices belong to a region are possible connected to multiple gateways. Given a device D_k, it has probabilities of $\{p_1, p_2, p_3\}$ to be assigned to gateways $\{G_1, G_2, G_3\}$, each of which is randomly chosen from all 10 gateways in the zone. In this study, we use $\{0.5, 0.3, 0.2\}$ as the gateway assignment probability.

In this study, we compared our proposed mapping algorithms with two other greedy algorithms without considering optimality (i.e., Static, Uniform). These algorithms include:

- Static: always greedily choose the first available device to host a component.
- Uniform: choose a gateway with a uniform distribution, then find a available device under the selected gateway to host a component.
- Optimal: the ILP algorithm for problem P_1 guarantee the optimality of reprogramming latency but doesn't consider run-time latency.
- Constrained: the IQCP algorithm for problem P_2.
- Hueristic: the heuristic algorithm that relocates service after round-up algorithm

We have studied the performance of our algorithms in terms of optimality of reprogramming time, which is represented by max number of reprogramming device under each gateway in a congestion zone, and run-time end to end deadline miss ratio.

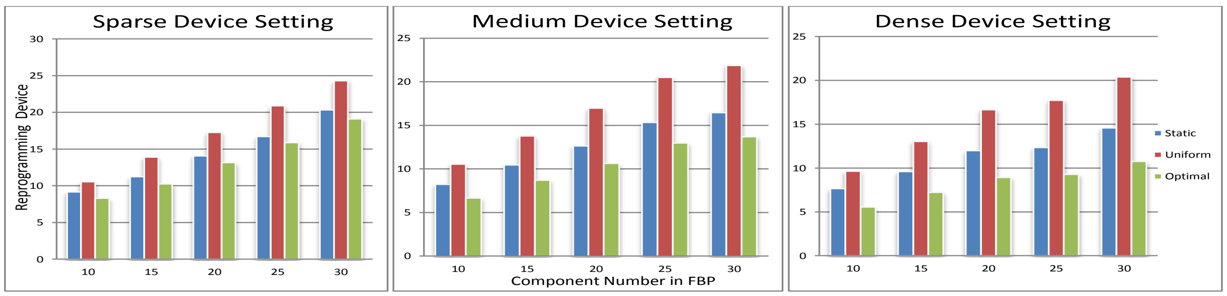


Figure 4.5: Device Setting Impact for Optimality

4.4.2 Performance of Problem P_1

We compare the Optimal algorithm with Static and Uniform algorithms to show its performance gain in terms of reprogramming latency. In Figure 4.5, every algorithm (static, uniform and optimal) gets better result with less number of devices to reprogram, as device setting changes from sparse to dense. In medium and dense setting, optimal much better than uniform algorithm, for uniform algorithm always uniformly choose gateway to host components, thus reducing the probability of collocate neighbor components. Static algorithm is comparable to optimal algorithm in sparse setting(less than 10 % penalty), for static algorithm increase the probability of collocation by choosing the first (sometimes the same) device to host two adjacent component. In medium and dense setting, optimal algorithm has 15% to 20% gain rather static algorithm, which means denser setting of device better improvement the optimal algorithm may achieve.

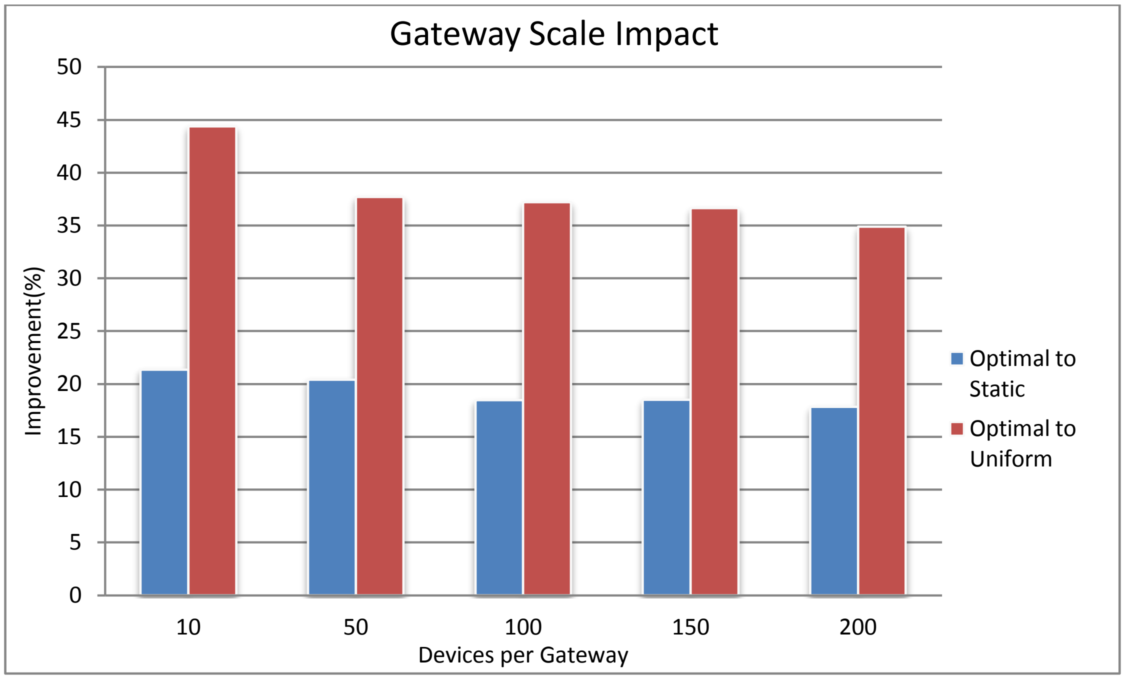


Figure 4.6: Gateway Scale Impact

In Figure 4.6, we report the impact of gateway to the improvement of optimal algorithm. In this study, we enlarge the number of devices under each gateway from 10 - 200, which means the total number of both regions and reprogramming copies increase from 10 to 200. As the scale of gateway increase, the improvement of optimal algorithm to static keeps larger than 18%. The reason behind this phenomenon is that more and more region are within only one gateway as gateway increases its size. The optimization problem consider whole zone is actually reduce to optimization problem on small regions within one gateway, thus the improvement ratio doesn't have obvious change.

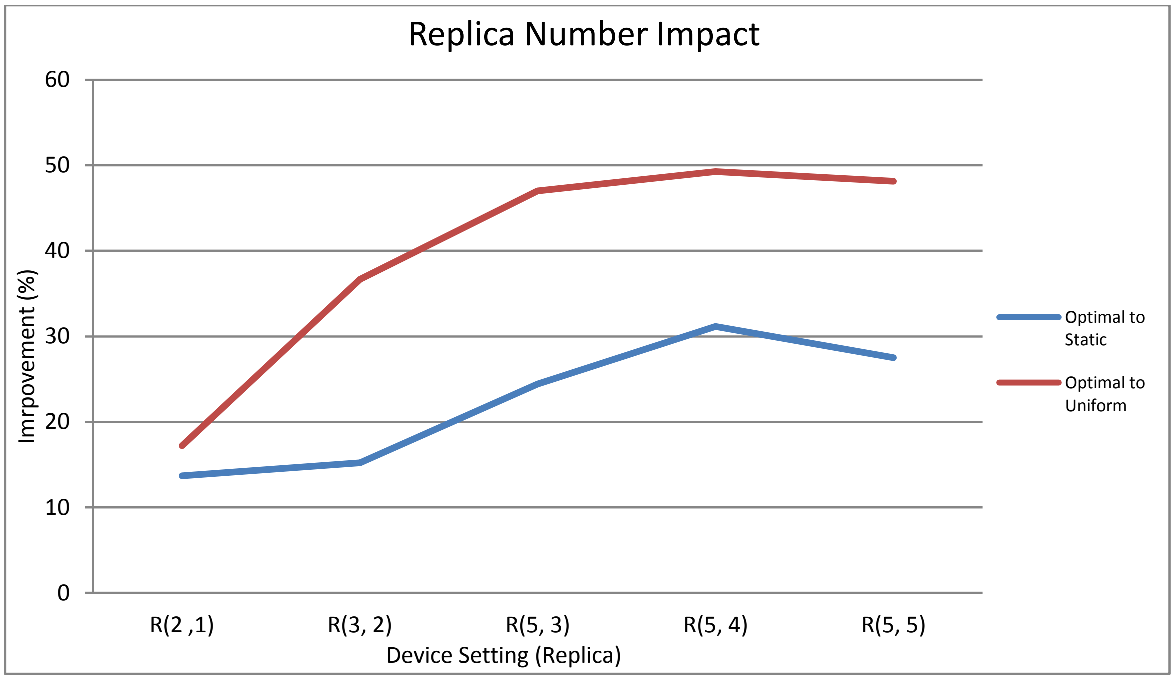


Figure 4.7: Replica Number Impact

In Figure 4.7, we demonstrate the impact of replica coefficient to improvement of optimal algorithm. The improvement ratio to both algorithms grow as replica coefficient increase from 1 to 4, but they go down when replica number becomes 5. It means more replication doesn't guarantee better performance of reprogramming latency. We also find that the improvement ratio grows every fast when replica number changes from 2 to 3, which means the best of setting is 3 replica in terms of return/cost ratio for fault tolerant critical environment, such as smart factory.

4.4.3 Performance of Problem P_2

In the second set of our experiments, we study the effectiveness of our heuristic solution for problem P_2. Our heuristic solution is obtained by the improved round-up algorithm for problem P_2 followed by a relocating procedure. We firstly show the penalty of heuristic algorithm in the aspect of miss run-time execution deadline, then shows its performance of achieving best trade-off between reprogramming latency and run-time cost. In this group of study, we use DAG shaped FBP with 20 components, the maximum runtime hops H used is 10, and target system includes 100 zones. Each zone has 100 devices and 10 gateways. The reason behind using relative small system scale is that constraint algorithm is intractable in bigger zone set.

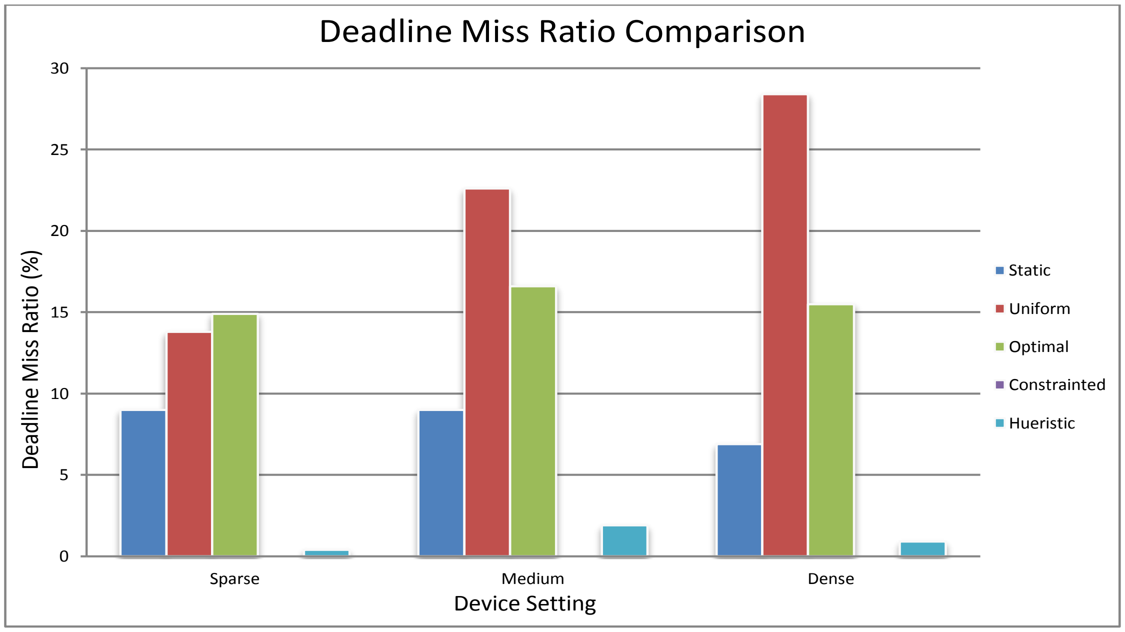


Figure 4.8: Deadline Miss Ratio Comparison

To evaluate the performance of each algorithms in terms of guarantee run-time latency, we firstly use the constrained algorithm to identify tractable problems (problem with solution that meets run-time deadline constraint), then run other algorithms to collect their deadline miss ratio. In Figure 4.8, our heuristic algorithm only has 1.5% deadline miss penalty, which is much better than static (6% - 8%), optimal without constraints (15%), and uniform (14% - 28%).

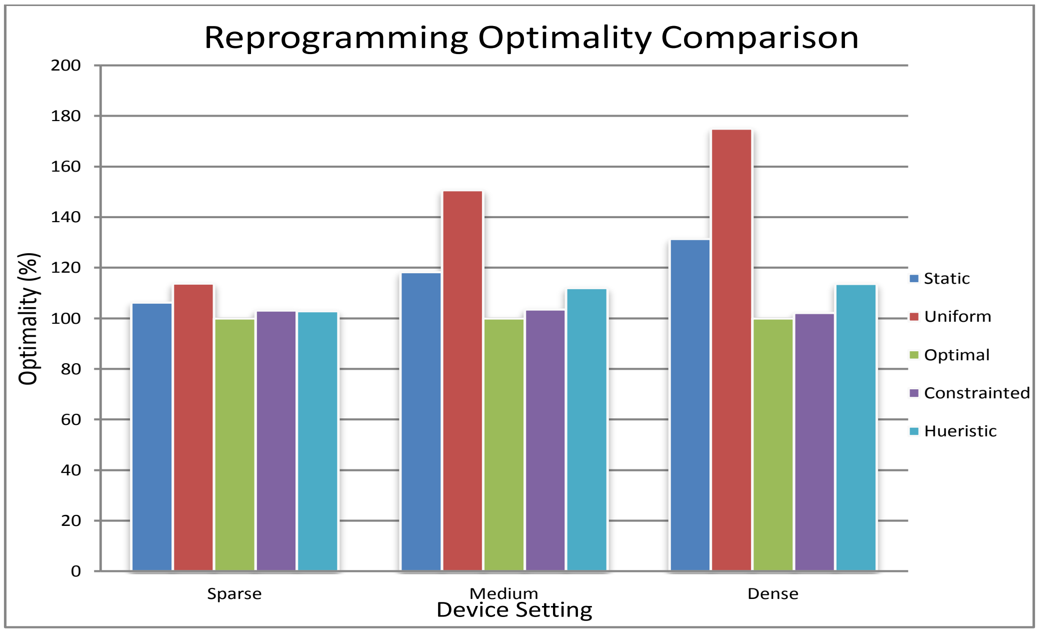


Figure 4.9: Reprogramming Latency Comparison

In Figure 4.9, we compare the reprogram latency optimality of each algorithm by setting the optimal algorithm as baseline. Our heuristic algorithm has less than $3\%, 8\%$, and 10% reprogramming penalty in sparse, medium and dense setting, which is still better than static and uniform. As we state in Section 3.1, the reprogramming latency is determined by two major components. One is mapping execution time T_{map}, the other is the code propagation time T_{m3}. In Table 4.1, we list the T_{map}, T_{m3} of each algorithm. We may found the optimal algorithm has the shortest time 40104 ms. Our heuristic algorithm is 43171 ms, which is better than any other algorithm. Even though the heuristic algorithm use about 2 more seconds to reprogram devices, but it is still the best solution for it small deadline miss ratio (less than 1.5%).

Algorithms	T_{map} **(ms)**	T_{m3} **(ms)**	**Sum (ms)**
Static	32	45466	45498
Uniform	39	61871	61910
Optimal	1729	38375	40104
Constrainted	27293	38623	66016
Hueristic	2012	41159	43171

Table 4.1: Time Distribution

4.5 Summary of Scalable Deployment

This section presents an heuristic algorithm for selecting and deploying flow-based IoT applications on large scale IoT environment such as smart campus. To achieve the low reprogramming latency, we model things into hierarchical regions, and use distributed solution to perform mapping decision and reprogram devices through gateway. Since low reprogramming latency and run-time deadline are two most significant factors of success of IoT applications, we have designed a mapping strategy from application to devices, that will trade-off both reprogramming time and run-time latency. We present a heuristic algorithm that round-up output of LP relaxation followed by a relocating strategy. Our simulation study shows that our proposed solution can use only 1/n * 0.7 reprogramming time than a centralized solution to reprogram a system with n congestion zones. If each zone is small, constrained algorithm can be used for guarantee both programming time optimality within run-time deadline. If each is large (more than 100 devices), we may use heuristic algorithm firstly for each zone. If any result of a zone miss deadline, we can further use constrained algorithm to find the better solution, for the deadline miss ratio of our heuristic algorithm is 1.5% which is relative small. In this way, we can guarantee an efficient deployment for most of regions in a system, at mean time erase the run-time deadline miss of applications.

Chapter 5

Edge Intelligence Support

5.1 Overview

The Internet of Things (IoT) envisions a world where billions of devices at the network edge gathering data and enacting commands to create innovative and intelligent systems. The new business opportunity drives companies promote their IoT platform and devices for edge computing. IBM announced the self-powered IoT starter kits based on popular Raspberry Pi board for building smart hospitals, homes, airports, etc. More recently, Microsoft and Amazon launched their own IoT platforms, which are Azure IoT Hub and AWS IoT. Both of them treat edge devices as data ingestion systems that collect, filter, and process received data before sending them to the cloud.

During this wave of evolution, dedicated, single purpose devices will give way to smart, adaptive devices that virtualize capabilities using a platform or API, collect and analyze data, and make their own decisions. Moreover, intelligent applications can be developed using such IoT platforms, for sensing and collecting information about our needs, then composing and deploying services to make our lives easier. In this chapter, we introduce the edge framework as part of WuKong middle-ware. This framework pushes the streaming processing capability from cloud onto edge devices, and provides reliable streaming analytics support to simplify the programming of intelligent IoT applications. We provide a simple and annotation based programming API for developers to implement online learning capability in the edge server. With such an extendable design, an intelligent component may be used for composing a flow based program (FBP) as an intelligent application.

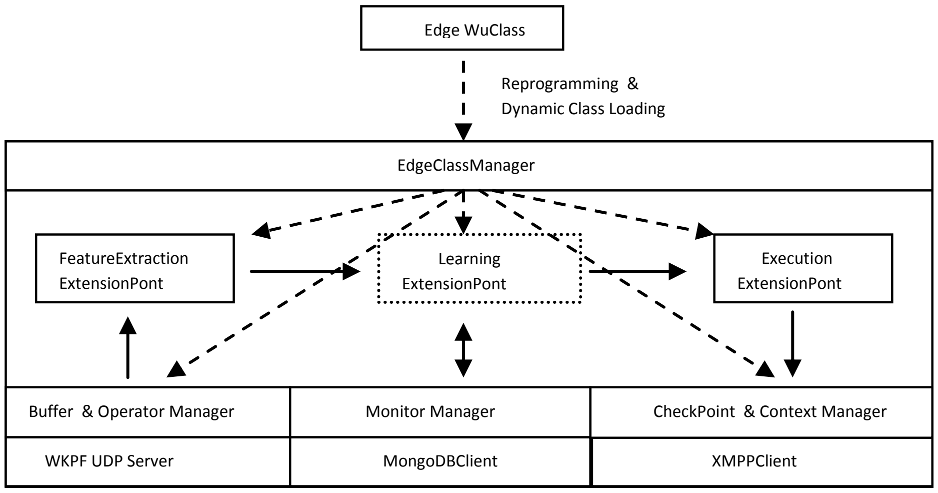


Figure 5.1: Edge Server Design

5.2 Edge Server Implementation

We have implemented edge server using Java, which a popular language for building streaming engines [5, 101]. In this section, we provide a high level view of the edge computing support including, programming model, native buffer design and extendable data pipeline. As shown in Figure 6.5, the edge server is designed as an streaming engine for processing IoT events. In the design, we mainly focus on providing a plug and play capability for building intelligent component.

5.2.1 Programming Model

We adopt the programming model of Web [31, 75] to achieve a high throughput and paralism. In the design, an intelligent component is called EdgeClass, which is a specialized WuClass. Intelligent component developers need to define input and output properties of an EdgeClass in the class definition. An EdgeClass will be loaded into edge server during application deployment through remote programming. Within the server, streaming processing scenarios are modeled as a soft pipeline, and implemented as event driven multi-stage data pipeline. We also provide a bunch of stream-transformation operators to choose features from data buffers within an edge server.

```java
public class EEGEdge extends EdgeClass {
  @WuProperty(name = 'raw', id = 0, type = PropertyType.Input, dtype = DataType.Buffer,
       capacity = 2000, interval = 1000, timeUnit = 30)
  private short raw;
  @WuProperty(name = 'output', id = 1, type = PropertyType.Output)
  private boolean signal;

  public List<Extension> registerExtension() {
    List<Extension> extensions = new ArrayList<Extension> ();
    extensions.add(new EEGFeatureExtractionExtension(this));
    extensions.add(new EEGExecutionExtension(this));
    return extensions;
  }
}
```

As shown in the code snippet above, the wuclass defines property raw as its input and property signal as its output. These two properties may be used in FBP for connecting with other WuClasses. When an edge server receives remote programming DJA file and identifies that the target application wants to use the EEGEdge class, the edge class manager will load the EEGEdge class into JVM. During class loading, the edge class manager will look up all of the annotations on each field through reflection. For example, the raw property is declared as data type buffer. Thus, the edge class manager will create a buffer for the EEGEdge object to hold the raw brain wave data sent from EEG device.

Besides declarative properties, the extension oriented interface is also an important feature in the design of EdgeClass API. As shown in Figure 6.5, the core of edge server is the extendable pipeline. An EdgeClass may define what to do in each stage of the pipeline by using the extension interface. For example, the EEGEdge class defines EEGFeatureExtractionExtenion and EEGExecutionExtension. During remote reprogramming, these two funtional programming components of an EEGEdge object will be binded to the extension point on the data pipeline respectively. During runtime, the brain wave data will be kept in the buffer, then flow into each stage of the data pipeline.

5.2.2 Networking

The edge server need to be discoverable by master, and be able to communicate with other devices in a WuKong system. To support cross network communication, Multiple Protocol Transport Network (MPTN) is designed to be distributed messaging gateways to enable messaging among multiple networks in WuKong middleware. As shown in Figure 5.2, MPTN is in the presentation and session layer, so that the routing capability in existing network protocols, such as Zwave and ZigBee can be reserved. MPTN gateway converts an end-to-end message request to multiple segment message based on network topology. The protocol schedules periodic routing table update to pro-actively keep the routing table up to date. The mechanism hides the complexity of heterogeous network environment in IoT systems, and allows on-demand route update to shorten the delay of cross network communication.

From the perspective of networking, the edge server is implemented as a UDP device in WuKong. When an edge server starts, it will firstly talk with a UDPGateway to ask for a

MPTN ID. After that, it will receive and send MPTN message from and to other devices in the network. In the application layer, the edge server also implements the WKPF interface, so that its capabilities (in terms of edge object) may be discovered by WuKong master. Once master deploys an application that uses an edge object in an edge server through reprogramming, the selected edge object will start to run.

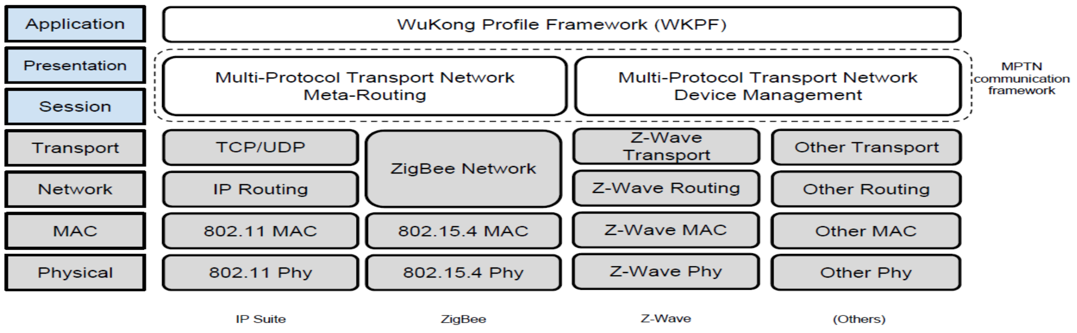


Figure 5.2: MPTN Architecture

During runtime, one single IO thread will receive MPTN messages from the gateway to which it connects. The IO thread use the Java NIO select API to fetching the MPTN data from channel buffer, and push the MPTN message into a ring buffer. A thread pool of workers repeatively pick up a message from the ring buffer, parse the MPTN and WKPF headers, and put the data in payload into the declared data structures in buffer manager. After that, it is the threads in data pipeline that are responsible for processing data in buffer manager.

5.2.3 Analytics Support on Streaming Data

In offline learning, the tedious tasks of ETL(extract, transform, load) are mandotary work before feeding into learning engine to train model. The main goal of edge framework is to push the data preprocessing from offline data pipleline into edge, so that the both the processing latency and network traffice may be reduced. Thus, edge framework aims to provide analytics support, and an efficient yet extensible data processing workflow for implementing self-learning capability in edge.

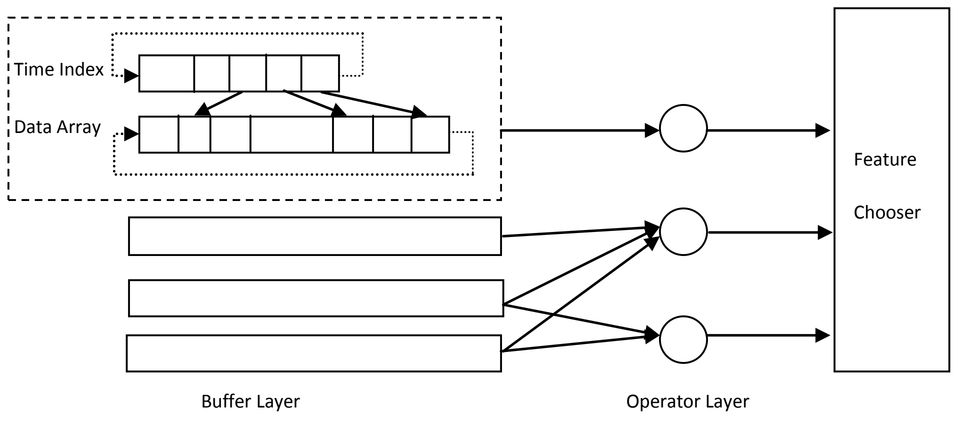


Figure 5.3: Double Ring Buffer and Feature Extraction

Compare to big data oriented distributed streaming systems [100, 101, 5], which are mainly to achieve scalable counting and ratio calculation on large scale logging event. The streaming analytics support of edge server is to process sensors and actuator datas in timely and memory efficient way. Thus, we have designed three type of data structures to meet QoS requirement of different application scenarios.

5.2.3.1 Memory Efficient Data Storage

For memory intensive applications, such as application of activity pattern discovery[25], the memory efficient design is desired for increasing the number of concurrent edge objects that a progression server can host. In the Oracle 64-bit HotSpot JVM, the header spaces for a regular object and for an array take 8 and 12 bytes, respectively. In a typical intelligent application, such as activity recognition, the heap often contains many small objects (such as Integer representing device ID, and Long presenting time-stamp), in which the overhead incurred by headers cannot be easily amortized by the actual sensor value. Therefore, we merge and organizing related small data record into few large objects (byte buffer) instead of representing them explicitly as one-object-per-data-point, and manipulating data by directly access buffers (Operators operate on byte chunk level as opposed to the object level). In this way, we bound the number of objects application, instead of making it grow proportionally with the carnality of the input data. Our double ring buffer and operators are built by strictly following the design paradigm.

- Real-time Channel It is implemented as simple pub/sub model in progression sever. When an edge object is initialized by master through reprogramming, edge server will create a channel for each input property annotated as channel.
- Double Ring Buffer Ring buffer is widely used for buffering data streams. To help efficiently query data within a period of time, we added a simple layer of index on the data array. The time index is also implemented as a ring buffer. Therefore, the special data structure for supporting operator based feature extraction is called Double Ring Buffer (DRB). With this data structure, data can be quickly fetched in dimension of both size and time interval from buffer.
- Time Series Operators A set of native operators have been implemented for quickly compose a feature extractor in an EdgeClass. Single input operator, such as Avr, Sum and Kalman-filter, accept data from one buffer. At the mean time, multiple input operator is pretty useful for join and cross filtering data in multiple sources. Window-able operators target mainly to the applications that similar to activity analysis, which the processing is triggered when the number of events arrives hits a threshold.

```
public class EEGFeatureExtractionExtension extends FeatureExtractionExtension<
    EEGEdgeClass> {
  @Override
  public List<Operator<?>> registerOperators() {
    List<Operator<?>> operators = new ArrayList<Operator<?>> ();
    RelativeIntensiveRatioOperator psi = new RelativeIntensiveRatioOperator();
    psi.addDataSource(0, 5);
    operators.add(psi);
    return operators;
  }
}
```

In the code snippet above, the EEGFeatureExtractionExtension is defined. Once the EEGEdgeClass is enabled in an edge server, the extension instance will be binded to feature extraction extension point in data pipeline through exposing the operator for extracting features from EEG brain wave raw data in a buffer. In this case, the relative intensive ratio operator is used. The operator is applied for property 0, which is the raw property declared as buffer every 5 seconds. Everything for binding data source and specify processing interval is achieved by using the add source interface. If multiple operators are used, the features extracted will be arrange according to the same order of operators returns, and be dispatched to other extensions of the edge class. We provide a library of time series operators for feature extraction. EdgeClass Developers may also implement its own operator for special purpose.

5.2.3.2 Extendable Data Pipeline

Learning data pipeline is designed to create a highly extendable programming model for learning life-cycle. The study of online learning algorithm [21] contributes a lot to the pipeline abstraction. At the mean time, the famous map reduce [26] programming paradigm in big data bring the idea of functional programming in extensions. It is designed to faithfully preserve the programming model of WuKong middle-ware. As a result, a streaming processing module can be easily ingest data generated by diversified sensor and actuators.

As we discussed in Section 5.2.1, an Edge Classs IO is defined in itself as property, but its real processing logic is defined seperatively in three types of extensions. which are shown in Figure 5.4. Feature extraction extension is the place that can define how to extract mearningful features from time series data buffer through using operators. The learning extension is useful for online learning algorithm to train model from flow of extracted features. Execution extension can be used for update the online models and use learning result to make configuration decision.

Within a learning pipeline, events are used to exchange information between extensions. To achieve simplicity, we also adopt the functional programming style to design the api of extension. During runtime, each event of learning pipeline will be dispatched to particular extension of an Edge Object, then corresponding function will be triggered.

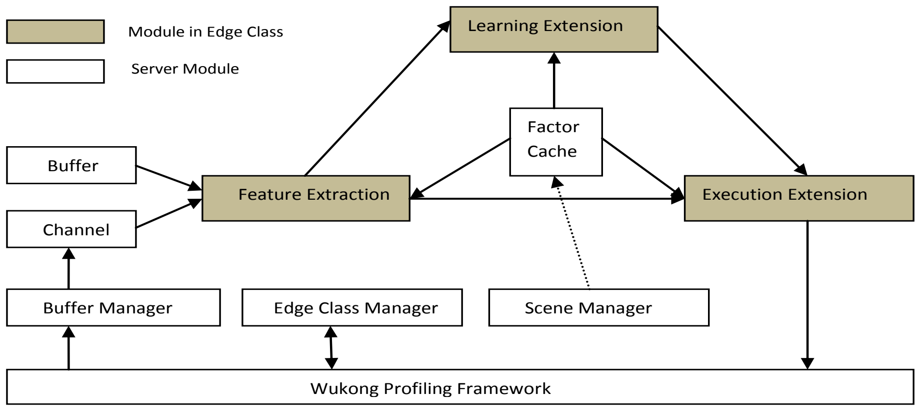


Figure 5.4: Extendable Data Pipeline Structure

Since learning life-cycle is diverse for different algorithms and applications, the progression pipeline is designed to be highly extendable. Pipeline is composed of multiple stages. Each stage is an extension point to which a subclass extension instance of a component can be registered. Within each stage, developer can define their own business logic, such as feature extraction, periodically learning, online classification, etc. For a learning scenario go beyond regular three stages, it is convenient to add new extension point at the right position on the pipeline. For example, boosting method are widely used in machine learning to improve accurary of prediction by binding output of multiple learning algorithms. To support boosting, the data pipeline may be extended as one feature extension point connects to multiple learning extension points, all of which connect to the execution extend point. In this way, the result of each model will finially forwarded to the execution extension for final decision making.

5.3 Distributed Runtime of Edge Framework

As shown in Figure5.5, edge server, XMPP server, and data store are new building block of edge framework compare to original Wukong middle-ware. To setup a Wukong cluster, all of the Wudevices and edge devices need to be registered in Wukong master. The multiple protocol transmission protocol (MPTN) [85] helps to register device with different kind of wireless protocol on the fly. For each Wudevice, WuClasses (stateless component) are created in Darjeeling JVM as driver for sensor and actuators plugged in WuDevices. At the mean time, EdgeClasses (stateful component) is be loaded in edge server on edge device. Before mapping and deploying any streaming DAG, master will discovery the capabilities of each devices in terms of type and number of components through Wukong Profile Framework (WKPF). During deployment, master map and deploy logic component in streaming DAG onto physical objects through remote programming. During run-time, WuObjects and EdgeObjects communicate with each other, following the streaming flow described in DAG.

5.3.1 Fault Tolerant Support

To ensure resilience to faults, the framework provides a mechanism to recover the state of failed component. It is achieved by providing a general model check-pointing service. Within the intelligent component, if a property is declared as backup, its content will periodically publish to a pub/sub topic, and retained within pub/sub system. When a new component is loaded, edge framework will search the legacy publications in the checkpoint topic, and initialize the value of that property.

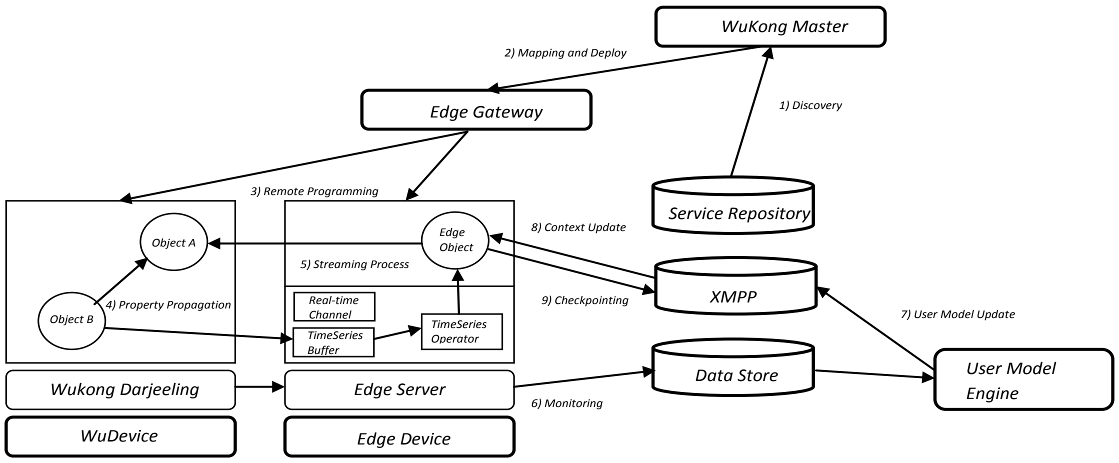


Figure 5.5: Edge Framework Run-time Overview

The fault tolerant consideration of edge server can be categorized into three types. The goal of its fault tolerant is to keep the correctness of control output, even when there are some physical device failure, and achieve fast recovery when the server itself failure.

5.3.2 Online Model Retention

Factors are the sign of external world, if an edge server fails, the online model learnt in memory will be gone. In a distributed edge instelligence architecture, which is of multiple edge servers, if an edge device physically fails, EdgeObjects on that particular device can migrate to another server in the system. In this case, we need have latest factor info stored externally. To achieve the goal, the edge framework utilizes the retention policy of pub/sub system. Therefore, latest 3 - 10 publications of a topic can be retained in pub/sub server, and be replayed later.

5.3.3 Model Checkpoint

The local model is the last but most significant data. Given an online algorithm, the local model is learned from a period of time, which is of most valuable parameters to control local environment. A checkpoint method is provided in edge framework to store the json format model info into a topic of pub/sub system. If master migrates an EdgeClass from one server to another, the new server can help to restore latest check pointed model back.

5.3.4 Context Engine Integration

In the edge intellignece framework, the integration between online learning and offline learning is achieved through global monitoring and pub/sub model update. If sensors and actuators which are chosen to monitor, all state updates are firstly collected in edge server, and is periodically bulk pushed to the data store. The offline engine periodically train the latest model from latest data in data store, and publish it through topic of XMPP. If an EdgeClass want to always fetch lastest model from that particular context engine, it just need to listen to the topic related with that context engine.

5.4 Edge Intelligence Case Study

With all of these facilities above, an IoT developer can quickly built an intelligent application on edge. In the case study, we show the development scenarios of an edge intelligence component in edge framework from data collection, through offline model learning to online classification in Edge Class. In the end, we continue the code snippets for EEGEdgeClass by introducing the main logic in execution extension.

Real-time Channel and Time Series Buffer and Operator are two important facilities built in edge server for developer to quickly build edge application. We build several edge applications to test evaluate our system performance. In this section, we introduce how to build brain wave control application by integrating with EEG (Brain Control Interface) BCI, and how to build indoor localization in detail. The first application demonstrate how to use our edge API to use facilities in edge server. The second one mainly to build for performance study in next section.

5.4.1 EEG Control

Over the past years, technology using electroencephalography (EEG) as a means of controlling electronic devices have become more innovative. Today, people are able to measure their own brain waves and patterns outside of medical laboratories. Furthermore, besides analyzing brain signals, these brain signals can be used as a means of controlling everyday electronic devices, which is also know as brain-computer interface. Brain-computer interface along with the "Internet of Things," are growing increasingly popular; more and more people have adapted to utilizing wearables and smart homes. In order to achieve this, we used NeuroSky Mindwave Mobile EEG, Raspberry Pi 2 with WuKong edge framework to build it from scratch.

5.4.1.1 EEG Background

From this active communication network of neurons, humans have derived distinguishable brain signals. The human brain is dynamic; it changes based on physical and mental activity. As a result, brain signals also change. These brain signals are categorized by different bandwidths and collaborate to describe human activity and function. We give a brief description of the most prevalent brain signals and their functions in the following section.

- DELTA WAVES

  The slow and loud Delta waves primarily exist between 0.5Hz and 3Hz. There are also known as deep sleep waves, since they are most prevalent when the human body is in a state of deep meditation or relaxation as well as deep sleep. During this period, the body is undergoing a process of healing and regeneration from the previous day's activities.

- THETA WAVES

  Theta waves are mostly generated around 4Hz to 7.5Hz range. These waves are associated with light meditation as well as sleep. When an increasingly number of theta waves are generated, the brain is in "dream-mode" state. In this state, humans experience the Rapid Eye Movement(REM) sleep.

- ALPHA WAVES

  Just above Theta waves are Alpha waves at 7.5Hz to 14Hz. Known as the deep relaxation waves, Alpha waves depict the resting state of the brain. These waves are dominant during a period of daydreaming or meditation. Alpha waves effect imagination, visualization, learning, memory, and concentration. They also aid in mental coordination, and mind-body awareness.

- BETA WAVES (12 TO 38 HZ)

  Beta waves exist mostly at 14Hz to 40Hz. These waves are associated with a person's consciousness and alertness. These waves are most prevalent when we are wide awake or alert, engaged in some form of mental activity such as problem solving or decision making.

- GAMMA WAVES

  From 38Hz to 42Hz, Gamma waves are the fastest and along with beta waves, are most prevalent when the person is alert and awake. These waves are associated with cognition, information processing, attention span, and memory. It is speculated the gamma waves can also denote a person's 'higher virtues', altruism, love, and spiritual emergence.

5.4.1.2 Offline Training Data Collection

We use an infinite while loop to continuously read data from EEG device. We define brainArray to be an empty list. This will act as a buffer to store all the data it receives from the EEG headset and refresh every five seconds. While in this time period, data is read continuously from the EEG headset and printed on to the console. We also use the sleep functionality from our imported time library to prevent overflow of data.

We specify an eyetag variable and set it either to 0 or 1. This is for testing and training purposes and to easily confirm whether the data we have on file is for eyes open or eyes closed. If the user has defined eyesOpen to be false, the eyetag is set to '0' and vice versa. Then we write the data to a single text file called brainArray-data.txt. We choosed three volunteers to collect offline learning data. For each of them, we collect 80 data points for each of eye-close and eye-opent status. Thus, we prepared in total 480 data points for our offline study.

5.4.1.3 Offline Training Classification Model

Before studying how to classify the data by using machine learning method, we firstly extracting the features from raw data collected from last step. We use the function of bin power in Pyeeg library [17] to extract the power spectral intensity of each type of waves. The graphs below demonstrate how a band of Delta, Theta, Alpha and Beta waves may distinguish eye-close and eye-open label.

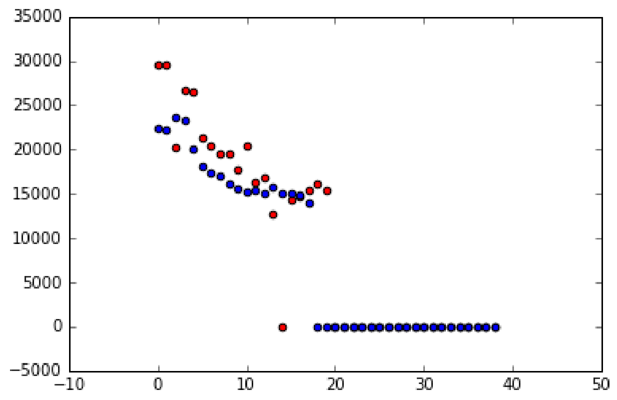


Figure 5.6: Delta Feature

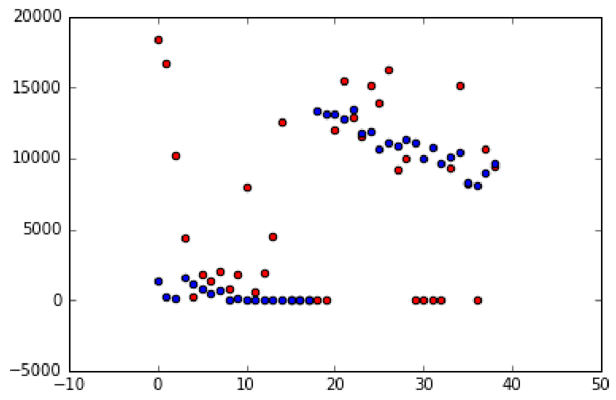


Figure 5.7: Theta Feature

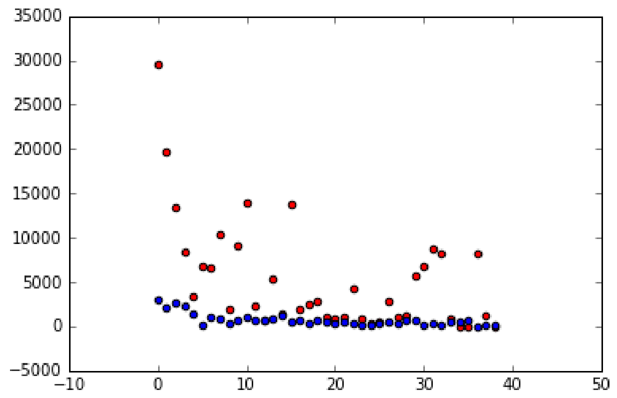


Figure 5.8: Alpha Feature

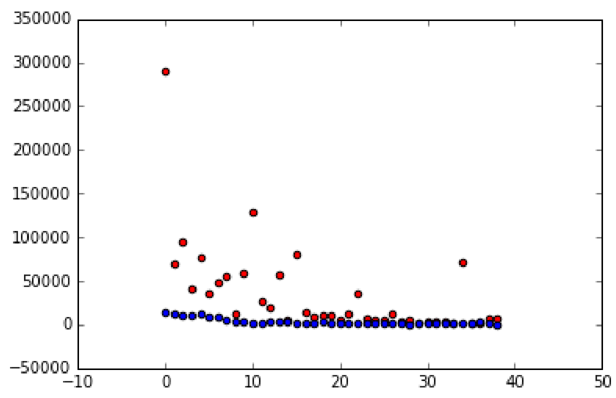


Figure 5.9: Beta Feature

The red points represent value of eye-close data points, and blue points represent value of eye-open. Intuitively learn from the diagram, we find that Alpha and Beta may categorize status of user very well. To gain best classification capability in run-time, we still study what's the accuracy of different classification algorithms under both of two features (Alpha and Beta), and all four features. During each round of study, we apply 10 folds cross validation in each validation.

As shown in Figure 5.10, we produced an accuracy of 61.8% and 56.8% using Ada boost and Support Vector Machine with Radial Basis Function Kernel respectively with Alpha and Beta. In Figure 5.11, it shows the result of applying all four features in classifiers. After

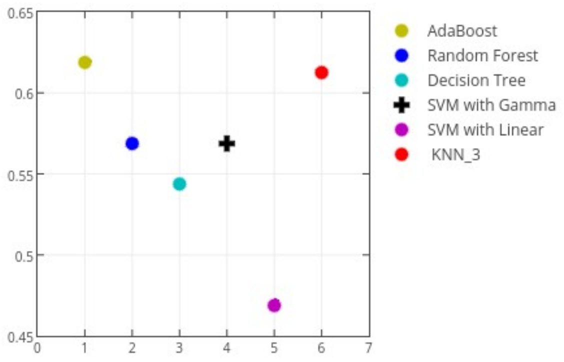


Figure 5.10: Accuracy using Alpha and Beta

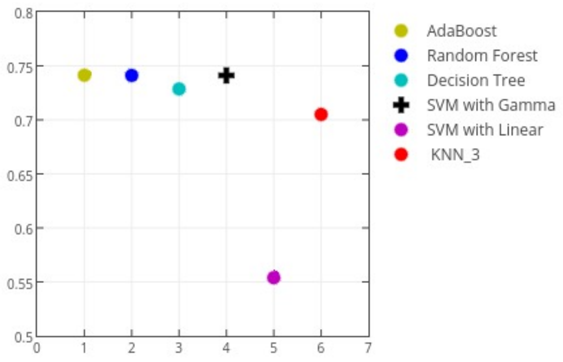


Figure 5.11: Accuracy using All

using all features, we produce an accuracy of round 88% by using support vector machine with Radial Based Function kernel. Because of the promising accuracy, we decided to use SVM as our model for the online study in eeg edge class.

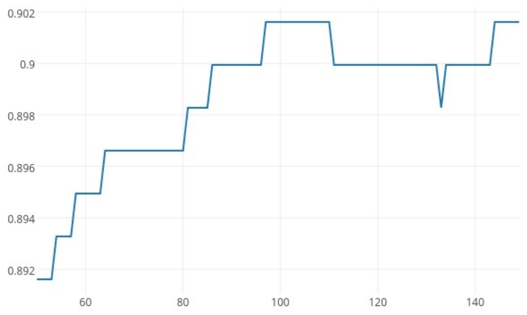


Figure 5.12: Parameter Gamma Impact

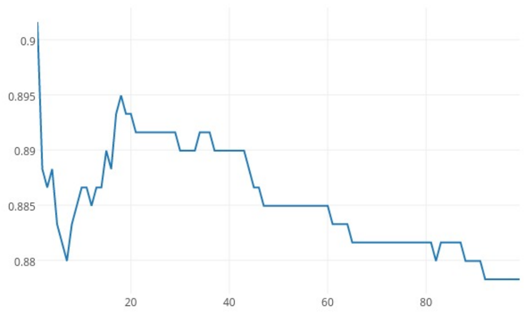


Figure 5.13: Parameter C Impact

After choosing Support vector machine with Radial Basis Function(RPF), we tried to tune two of the SVM RPF parameters, Gamma and C. Gamma is defined as how far the influence of a single training example reaches, with low values meaning open and high values meaning close. C is defined as trade-off mis-classification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors. After tuning these parameter using a binary search algorithm, we concluded the best Gamma = 106 and best c= 1. This will give us an accuracy of 90%.

After offline training and tuning model parameters through using data collected every 5 seconds. The last step is to use WuKong edge framework to connect EEG signal with the SVM model and control the physical LED light. In the last section, we introduce how to use WuKong application framework achieve inter-operation between EEGDevice, threshold and LED through using the FBP. Now, we want to replace the threshold with an intelligent component called EEG Edge Class to do the signal classification in real-time.

5.4.1.4 EGG Control

In the EdgeClass, the input property is declare as a buffer (DoubleRingBuffer), whose data ring capacity is 2000 data points, index ring capacity is 30 units, and index is built every 1000 milliseconds. Therefore, the buffer will hold data in a time window of 30 seconds, it will at most keep 2000 data points. The buffer will store raw signal from EEGDevice from which time series operators will fetch data and generate features.

```java
public class EEGExecutionExtension extends AbstractExecutionExtension<EEGEdgeClass>
    implements Executable<Number>, Initiable {
  private static Configuration configuration = Configuration.getInstance();
  private static Logger logger = LoggerFactory.getLogger(EEGExecutionExtension.class);
  private svm_model model = null;
  private double[] labelProbabilities = new double[2];

  public void execute(List<Number> data, ExecutionContext context) {
    if (data.size() == 4) {
      StringBuilder builder = new StringBuilder();
      svm_node[] nodes = new svm_node[4];
      for (int i = 0; i < data.size(); i++) {
        builder.append(data.get(i).toString());
        nodes[i] = new svm_node();
        nodes[i].index = i;
        nodes[i].value = (double)data.get(i);
      }

      double probability = svm.svm_predict_probability(model, nodes, labelProbabilities)
          ;
      // 0.0 represents close
      if (labelProbabilities[0] >= 0.88) {
        logger.info("Set output to true, when eye close");
        this.edgeClass.setOutput(true);
      } else {
        this.edgeClass.setOutput(false);
      }
    }
  }

  public boolean init() {
    try {
      model = svm.svm_load_model(MODEL_PATH);
    } catch (Exception e) {
      logger.error(e.toString());
      logger.error("Fail to initialize svm model");
      return false;
    }
    return true;
  }
}
```

Since model are offline trained in the EEG study, we ignore the online learning extension, and only focus on how to use model to do online classification on the features extracted. Below is the implementation of the major business of logic of intelligence. Firstly the capability of execution extension can be expanded by implementing more interfaces. Here, the EEGExecutionExtension implements both Executable¡Number¿ and Initiable. Within init function, we firstly load the model from local file system. The model is generated by libsvm on the trainning data and tuned SVM parameters (C and Gamma). The execute function accept features in a list as the first parameters, and execution context as second parameter. Within the function, we use the model to classify whether should label the features as eye close or eye open. We set 0.88 as the probability threshold to trigger eye close action by setting the output value. We tested the application on real physical devices. Its demo can be found in the https://www.youtube.com/watch?v=E0U9MoJzxoo.

5.5 Performance Study

5.5.1 Scalability

We study system scalability in three dimensions: scale in QPS, scale in number of Edgeobjects, and scale in memory efficiency. Since activity recognition is aperiodically triggered and occupancy prediction is periodically triggered, we use the localization application to evaluate the scalability of edge server. In the localization EdgeClass, the predefined map has 100 * 100 siganl areas. Initially, we create 1000 filters uniformly distributed on the map. The normal execution time of the localization algorithm in Raspberry Pi is about 37ms.

Figure 5.15 shows how single localization component performs in a edge server, when we increase the rate of localization request. As we increase the qps (query per second) from 2 to 24, the response time is almost linearly increase from 80 ms to 240 ms. After 24 QPS, the

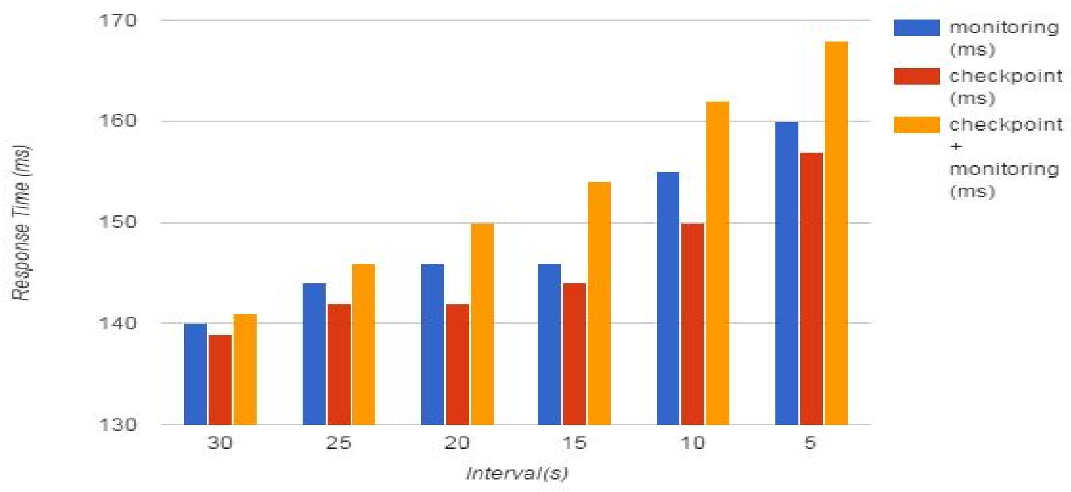


Figure 5.14: Monitoring and Checkpoint Overhead

response time start to exponentially increase to more than 1500 ms at the rate 32 qps. If we assume the reason response time of indoor localization is about 1 second, we may conclude that the maximal throughput of localization on Raspberry Pi 2 is about 30 per second.

Beside the single edge object throughput, we also evaluate how average execution time and response time changes, as we exponentially increase the size of edge objects hosted in a single edge server. In Figure 5.16, we may find that there is some acceptable extra cost (less than 30 ms) for both localization and activity recognition's execution time. The overhead is introduced by extension management in the software pipeline, and event dispatch for each edge Object.

Beside the setting of single server on board, we also investegate the potential performance gain of multi-core in Respberry Pi 2. In orange line shows the execution time of edge objects evenly split into two edge servers, which are binded to two different set cores by using Linux taskset tool. We find two JVMs competing with I/O and CPU resource, and increase the overhead of process level context switch in the setting. As shown in Figure5.16, single server setting always surpass the setting of two servers.

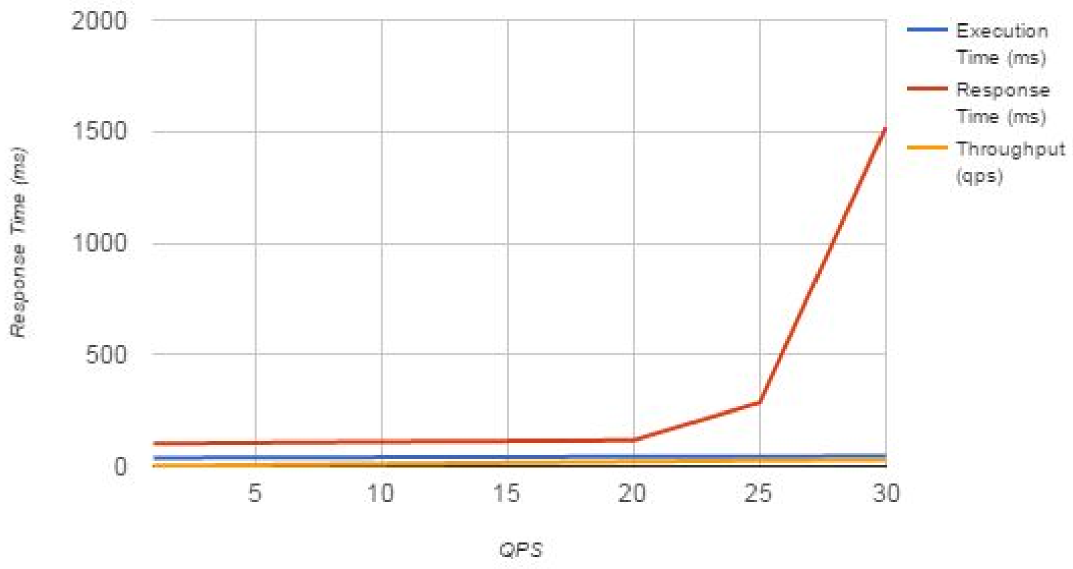


Figure 5.15: Localization Throughput

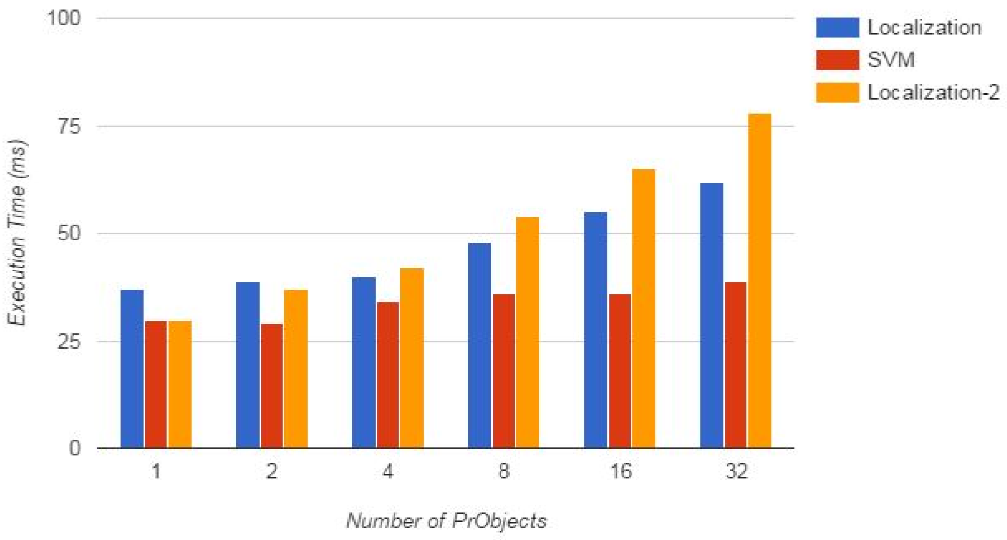


Figure 5.16: Scalability in Execution Time

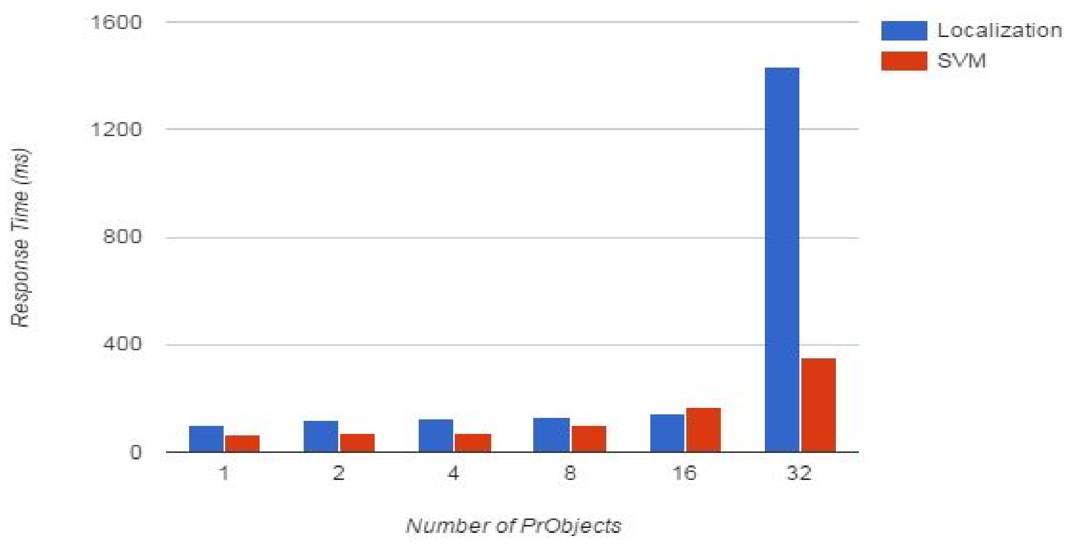


Figure 5.17: Scalability in Response Time

Figure 5.17 gives the same hint of Figure 5.15. As we increase the number of localization edge object to 32, the response time rapidly increase to around 1300ms, which means 28 - 30 localization edge objects is the maximal load a progression server can handle with.

For memory efficiency, we mainly study how our ByteBuffer oriented design can save memory, comparing with Java object oriented design, in the activity pattern discovery [25] scenarios. We use the CASAS [7] Kyoto Daily life (Spring 2009) dataset. The raw data contains 1299775 data points. We gradually feed all sensor values into two Probjets, and keep monitoring the memory usage of each of them. Excluding the initial 12MB memory usage when the server starts, the Java Object array list based edge object use about 6MB to keep all of data in memory. Since the limit memory in edge device, save memory usage for stateful components will increase the number of PrObjects that can be hosted, thus increase the scalability of whole system.

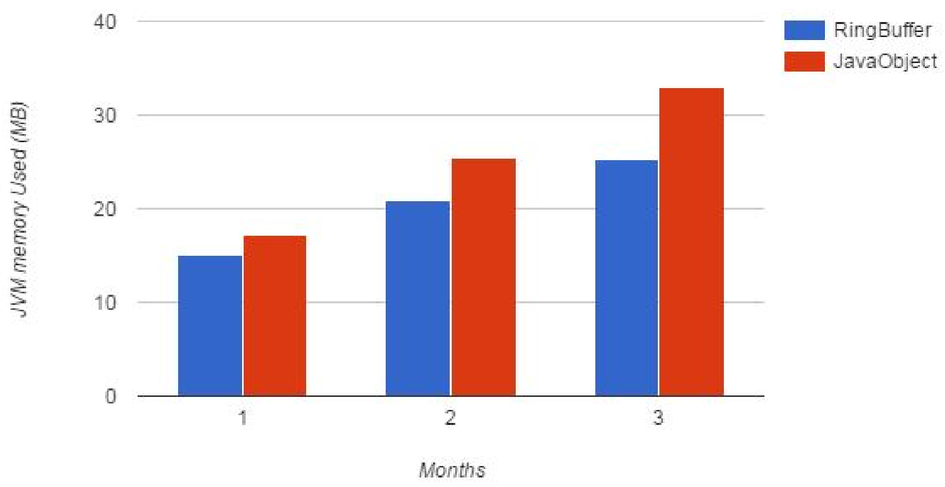


Figure 5.18: Scalability in Memory Efficiency

5.5.2 Fault Tolerant

As we discussed in Section 5.3.1, we provide the fast recovery of internal states EdgeObject through check-pointing the model as Pub/Sub messages. Once we find a failure of edge object caused by an edge server, the master will detect the failure and migrate the edge object to another server nearby. Figure 5.14 shows the overhead of monitoring and checkpoint for fully loaded edge server, in which we run 28 localization edge object and one activity recgonition edge object. In the experiment, all of the wifi signal to 28 edge objects are monitored, and only the activity recognition edge object checkpoint its SVM parameters. You may find that as we decrease the interval, the average response time increase accordingly. Since it is almost a fully loaded situation, it means the overhead is acceptable in normal setting. The recovery time of a stateful component includes fault detection, link update in link table and checkpoint replay. The svm model in activity recognition is used for evaluation, and the heartbeat internal is set to 2 seconds. It takes around 3.5 seconds to recover the execution. Given any smart home application scenarios, it is adoptable.

EdgeClass	Source Code Lines	Size of Byte-code
Localization	879	5.3KB
Activity Recognition	378	2.3KB
Occupancy Prediction	302	1.8KB

Table 5.1: EdgeClass Code Size

5.5.3 Reconfiguration Overhead

To ensure the correctness of reconfiguration, the two phase commit protocol is implemented for updating both the transceiver and receiver of a streaming link. In low bandwidth wireless network such as Zwave, the round trip time is about 100 ms in light traffic, and 400ms in heavy traffic. Since management component update links one by one, thus the overall reconfiguration time for a partial DAG is linear to the number of links N need to be updated, which is about *100N* ms to *400N* ms.

5.5.4 Programming Simplicity

To evaluate the gain of programming simplicity, we have implemented three types of common applications in smart home environment. Their code size is listed in Table5.1. For activity recognition Edgeclass, we use the libsvm library for SVM classification, so its source code base is not that big. Nevertheless, we implement our particle filter algorithm in indoor localization EdgeClass, so its code size is biggest. The merit of Java helps us to dynamically loading new EdgeClass, which means new functionality can be added during run-time. At the mean time, with the streaming support of edge framework, an intelligent application can be quick build up without considering the problem of accessibility to sensor and actuator on different hardware platform, connectivity to diversified network protocol, and failure recovery. With the support of Wukong middle-ware, A streaming DAG can be easily deploy to target edge environment. These benefits will magically increase the productivity of edge application development.

5.6 Summary of Edge Intelligence Support

We are witnessing the emergence of a new class of embedded IoT applications that involve continuous local analysis and also utilize the model from learned from big data for intelligence. This chapter presents a system architecture, the edge intelligence framework, that takes on the challenge of building a distributed system infrastructure for reliable low latency intelligent application in IoT. As the most significant component of edge intelligence framework, edge server is devised for the purpose of build intelligent EdgeClass with simplified programming paradigm. We also discuss how to build EEG Control from scratch as an use case of edge framework. We believe this new type of IoT application programming methodology will drive the design of next-generation distributed IoT systems and intelligent applications.

Chapter 6

Progressive System Design and Implementation

6.1 Overview

Progression framework defines an open layer for policy driven runtime management in the WuKong middleware. For the system level progression, a Policy and PrClass developer needs to define a mapping algorithm and a PrClass to manage the runtime of applications. By this way, progression framework may extend the capability of WuKong to manage its hosted applications.

The concept of progression is defined as the process of improvement over a period of time through a continuous and connected series of actions, event, etc in Merriam-Webster dictionary. In the definition, it emphasizes the continuous improvement through a process. In this section, the progression model will be presented. After that the progression in both the system layer and the application layer will be discussed.

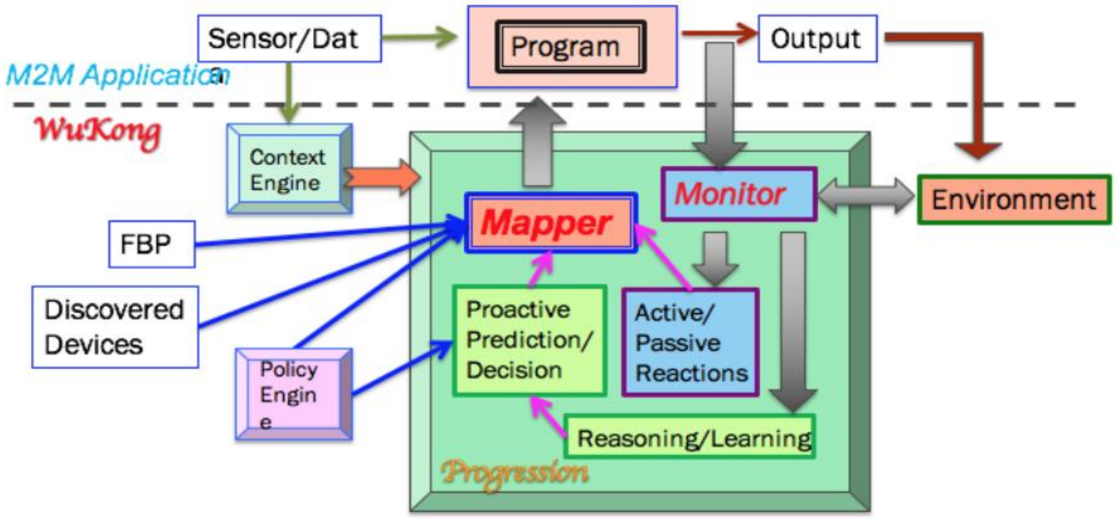


Figure 6.1: Progression Model

6.1.1 Attributes of Progression Model

The concept of Autonomic Computing [42] was proposed by IBM in 2001. The MAPE-K (Monitoring, Analyze, Plan, Execute, Knowledge) model is proposed to define the fundamental characteristics of autonomic computing systems, and a practical framework for supporting self-management and autonomic behavior in heterogeneous system environment. The vision of autonomic computing further was discussed in [54, 32]. Another model called 3D (Detect, Diagnosis, Defuse) was proposed for building accountability [104, 105] in SOA.

Both of MAPE-K and 3D model are based on offline learning, and passive reaction. More recent, IoT applications require online learning capability, such as occupancy aware preheating in smart home [83], activity sensitive energy management in smart office [67], HAVC control in smart building [28]. In these intelligent IoT applications, system are proactively adjusted based on online learnt patterns. To meet the requirement of gap between emergent needs and autonomic computing in IoT, the progression framework is designed mainly for providing a capability of proactive management for IoT applications in middle-ware layer.

Figure 6.1 demonstrates the progression model in WuKong system. There are two control loops in the model, which are reactive loop and proactive loop. Both of them need to take monitored information from execution of applications and environment as input. In summary, the progression model has the following major features:

- **Self-learning:** Progression model extracts useful control principle from online streaming data, additional to accumulating knowledge from historical data. It brings freshness and responsiveness of control into the IoT systems.
- **Policy Driven:** The proactive control is coupled with mapper through policy, which means the proactive control is to online tune resource parameters and optimize systems, according to the initial mapping decision. Beside this, it is an extendable framework for providing more autonomic management policy in IoT systems.
- **Context Aware:** Similar with knowledge in MAPE-K model, the context learned from historical data is applied to every aspect of progression. Thus, the context input may be applied to monitoring, learning, reactive/proactive control and mapping.

The idea of progression can be adopted in two different layers. One is the progression in system that supports an application to progressively maintain its healthy and performance; the other is the progression in application that keeps on tuning the serving model by mining latest pattern from user's behavior.

6.1.2 Progression in System

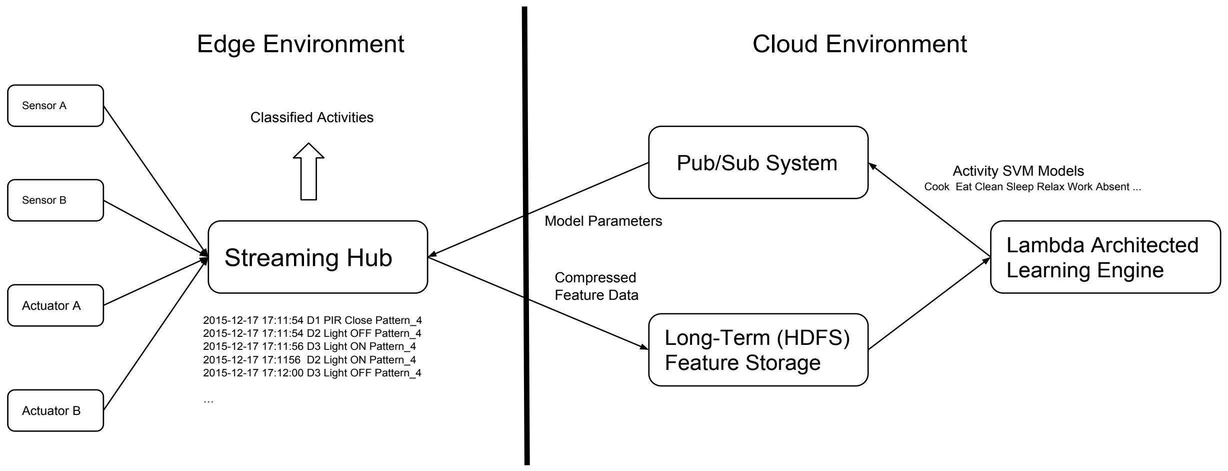
An IoT application in progression framework should be self-optimized. Given a FBP with N components and a WuKong system with M devices, we use variable x_{ik} to represent the decision whether to map component C_i to service S_{ik} on device D_k. Given a link L_{ij} of the

FBP, we use the set P(i, j) to denote the set of devices pair (D_p, D_q) tha can host link L_{ij}. If use parameter π_i denote the cost on component C_i, and parameter ω_{ipjq} to represent the cost of link L_{ij} (if map to device pair (D_p, D_q)), the generalized mapping problem can be defined as:

$$\min\left(\varphi\left(\sum_k (\pi_i x_{ik})\right) + \varphi\left(\sum_{(D_p,D_q)\in P(i,j)} (\omega_{ipjq} x_{ip} x_{jq})\right)\right) \tag{6.1}$$

Inside the formula, the first part is the cost on devices, the second part is the cost on links. In some cases, one of them can be ignored, according the requirement of policy (we usually don't consider cost on link for fault tolerant mapping). The function φ is replaceable for different policy. For example, if the goal is to minimize total cost, φ is the sum on all of possible objects, but if the goal is to maximize the system life time, φ should be the function of max. During run-time, these parameters will be updated because of run-time dynamics. Through the steps of auto-detection, intelligent diagnosis and efficient recovery, progression framework aims to make the IoT system self-optimized during run-time according to original optimization goal.

On the other hand, the goal of fault tolerance computing is to enable systems to continually operate properly. Major technologies to achieve fault tolerance, including replications, check-pointing and recovery lines, and transactions [89, 27], focus on recovery from failures. In our framework, fault tolerance in IoT is enhanced through automatic detecting failure of devices, and recovering the IoT application through reconfiguring the link table in Darjeeling JVM.



```
2015-12-17 17:11:54 D1 PIR Close Pattern_4
2015-12-17 17:11:54 D2 Light OFF Pattern_4
2015-12-17 17:11:56 D3 Light ON Pattern_4
2015-12-17 17:1156  D2 Light ON Pattern_4
2015-12-17 17:12:00 D3 Light OFF Pattern_4

...
```

Figure 6.2: Progressive Data Flow of Edge Intelligence

6.1.3 Progression in Application

As shown in Figure 6.2, the streaming hub functions as a gateway between devices in edge and system on cloud. In the design, we put the functionality of data pre-processing such as grouping, ordering and feature extraction in the hub, and leave high computation cost task, such as model training on the cloud. The models are propagate from cloud to hub through pub/sub channels and the classification task can be processed near to the data, thus reduce the latency for triggering actions for users.

Beside interplay of edge and cloud, the progression model also emphasize the continuous development of intelligence on edge, which means the interplay is not only a collaborative work, but also a mutual improvement. For example, the activity recognition as a classification problem usually need to perform feature extraction from sensor events that are buffered in a window. The classification model is trained on predefined labels on collected data, but on the edge the self-learning module may be used for discovering patterns in the data that does not belong to a predefined class. At the mean time, new generated label aids in understanding the latest data and segmenting it into learn-able classes in cloud. Thus, the run-time model in edge server can be always fresh in terms of reflecting latest pattern of users.

6.2 Progressive System Architecture

6.2.1 System Architecture

The system architecture of the framework is shown in Figure 6.3. To meet the scalability requirement of IoT systems, the architecture is divided into a subsystem in cloud and a subsystem in edge. Within an edge environment, the edge subsystems forms an local management zone. Thus, low level autonomic management tasks, i.e fault tolerant reconfiguration, online context driven system adjustment, may take place on edge without interfering the centralized master in cloud.

6.2.1.1 Subsystem in Cloud

In section 2.2, the original WuKong architecture has been introduced without inclduing edge and progression framework related components. Here, we elaborate the new components added for supporting progression in the new hierarchical management architecture.

- **Master:** Addition to original functionality of service repository, mapping, it is expanded to handle with progressive remapping by receiving feedbacks from managed progression server located in different management zone on edge.
- **Data Storage:** A centralized data storage that keeps historical user's behavior, sensor and actuator changes in database. It is implemented on top of MongoDB.
- **Context Engines:** Each of context engines periodically processes historical data, perform offline learning on the data set, and publish the context and knownledge to progression server or master. For example, virtual wuclass engine [107], and user preference engine [48] have been implemented in our framework.

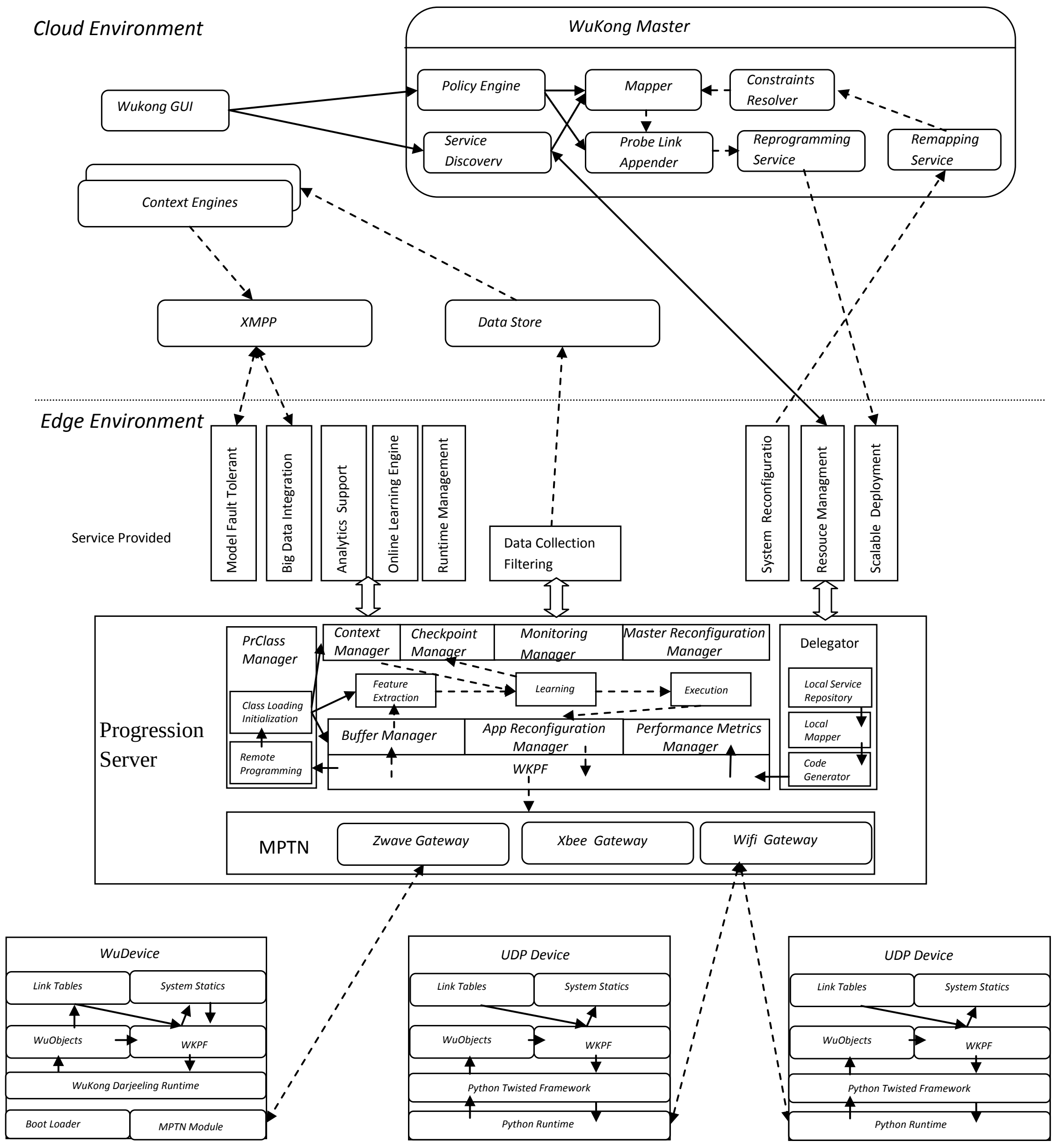


Figure 6.3: Progression Framework Architecture

- **XMPP:** For intelligent component with learning capability, XMPP retains the check-pointing messages for online models. For progression framwork, it function as a brige between engines and edge devices for exchanging knowledge and context.

6.2.1.2 Subsystem on Edge

As shown in Figure 6.3, the progression is achieved through cooperation among master, progression server, Wudevices. Within a management region, all devices are managed by a progression server during runtime. At the mean time, region delegator is responsible for scalable reconfiguration through reprogramming.

- **Progression Server:** It inherits from edge server introduced in section 5. Thus, it contains streaming pipleline for self-learning. Within the server, developer may develop progression class (PrClass) to processing the monitoring stream. The server provides native capabilities to reconfigure applications through extended WKPF, and send report to notify master to remapping and deploy an application. A PrClass may inspect the application status by looking into the environment status stream, and online adjust configuration of a running application into an optimal status according to specific mapping policy.
- **Region Delegator:** As introduced in section 4, scalable reconfiguration is achieved by using master delegators for making mapping decision and code generation. Region delegator is a service that is colocated in progression server on the same device. It is responsible for reprogramming devices in a management region.
- **Gateway:** It not only routes MPTN message cross different network, but also lookup monitoring messages and forward them to progression server.
- **WuDevice:** It runs WuKong darjeeling for controlling end sensor and actuators, and

it connects to Zwave gateway.

In the framework, each application deployed in WuKong is customized by the probe selector in master. The framework supports two types of probes, which are property probe and status probe. Property probes are specified by FBP developer during draw the FBP, and status probe are automatically added for the mapping policy selected by users. For each property probe, link append-er will add an additionally link in FBP, so that value update of property will be pushed to progression server automatically.

The probe selection strategy is customizable for status probe. Given a mapping policy, for example energy efficiency, the probe selector will add probe info into the DJA file before reprogramming. During run-time, each status probe functions as interceptor in WuKong profiling framework. During the execution of a WuKong application, WuKong Darjeeling and edge servers collect run-time statistics in memory, for example darjeeling collects number of messages sent out in each link for energy efficiency policy. These run-time statistics can be polled by programs running in progression server. Progression server, constraint resolver and probe selector in master, progression agents (interceptors in Darjeeling and Edge server) are the focuses in my dissertation.

6.2.2 Progression Lifecycle

The system lifecyle is shown in Figure 6.4. Progression frameowork has three work-flow loops, a big loop, a medium loop, and a small loop. The big loop includes eight steps: mapping policy choosing, service mapping, probe selection, append monitoring links, reprogramming, run-time monitoring, optimality deviation detection, remapping request. The medium loop includes four steps: optimality deviation detection, application reconfiguration, update link table, and run-time monitoring. If the optimality deviation needs a global adjustment the big loop reconfiguration will be applied; otherwise, the small loop reconfiguration will be ap-

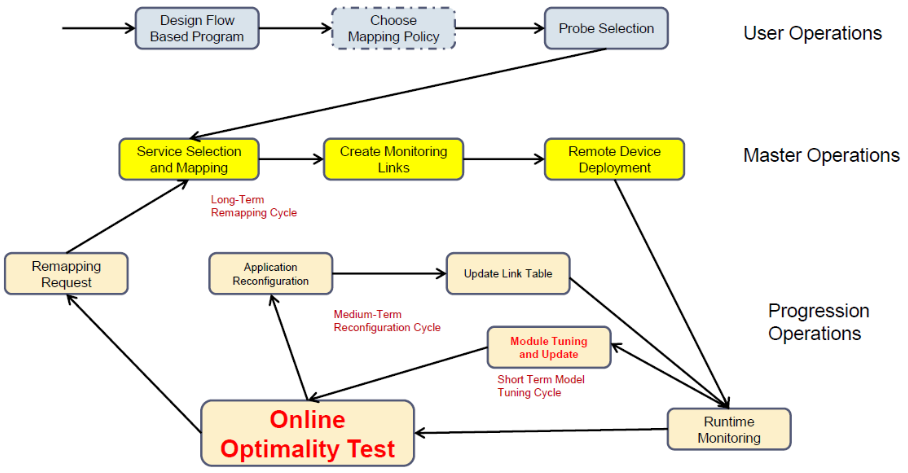


Figure 6.4: Lifecyle of Progression Framework

plied. The small loop only contains three steps: monitoring, offline learnt model tuning, and optimality deviation detection. These are most important four operations that progression framework provides for autonomically manage an application in WuKong middle-ware.

- **Online Optimality Test:** determine whether system or application is in an optimal status by looking into data stored internal in-memory data storage
- **Remapping:** The long term progression operation helps to rearch a global optimal status, but it will have a big delay or interruption of service in the system. It should be used predictively.
- **Reconfiguration:** The short term progression operation helps to reach a quick solution to recovery to a healthy system status. Even through it can't always achieve global optimal, but it incur much less service interruption than remapping.
- **Model Tuning:** The model for runtime decision makeing is updated through pub/sub. No service interruption are needed. Thus, system may smoothly evolve according to the environment fluctuation and context change.

6.3 Progression Framework Implementation

The progression framework consists of a set of monitoring interceptors in Wukong darjeeling JVM [8], progression server as host of progression component, and a set of services for reconfiguring the wukong ecosystem.

6.3.1 Intercepter in Darjeeling

As we discussed in section 6.2, there are two types of probes in the system. The property probe is monitored through appending monitoring link directly in FBP. Once the propety is updated, its value will be sent to progression server through property propagation routine in WKPF. The status probe is implemented as interceptor in WKPF. Once there is an operation happens in WKPF, the counters for the operation will be recorded in memory. These status can be pulled by PrClass through WKPF polling message.

6.3.2 Progression Server

As shown in Figure 6.3, the progression server is the driver of whole progression framework. It collects information from devices, perform online analyze on the status data, and notify master to remapping the application or reconfigure devices once an optimality deviation is detected. Since the progression server is similar as an edge server that needs to do streaming process, it inherits the merits of edge server with full advanced support in its implementation. Given one type of mapping policy, developer may devise a progression class (PrClass) according our programming API. As an application is deployed according to a specific policy, the corresponding PrClass is loaded into the progression server, and start to monitor and predicatively maintain the status of the application.

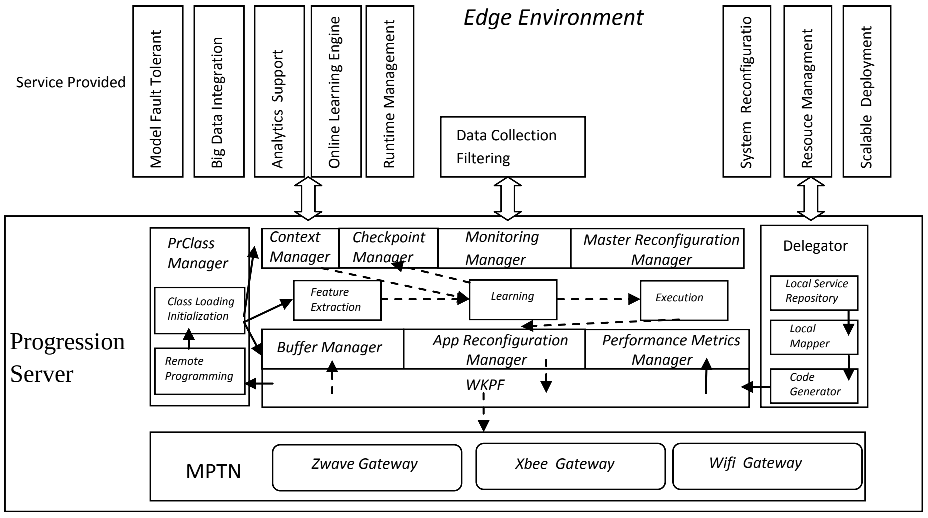


Figure 6.5: Progression Server Design

6.3.3 Optimality Test Support

Both of the progression server and the edge server are implemented as a streaming server, but the data they process is greatly different from each other. An edge class functions as an component in a FBP. It receives output streaming of precedent components and generate write property message to change status of subsequent components. However, a progression class need to collect information of whole FBP, or a category of information of whole system, for example collecting info of all of PIR sensors in a home to predict user's absent. According to this requirement, two new types of data structures are designed.

- **Global Channel:** It is a specialize channel. It may receive all of monitored message from the system. Each message in the channel is identified by network Id, port Id, and property Id (NPP) and WuClass type. It is the responsibility of channel user (extensions of PrClass) to distinguish whether to process a message or not.

- **System Buffer:** It is a specialized time series buffer. It keep historical time-series data that belongs to one particular category of sensors or actuators. For example, a PIR system buffer stores data of all the PIR sensors in the running system.

6.3.4 Progression Services

Within the progression server, there are a set of services for PrClass to adjust the system setting and configuration after optimality test.

6.3.4.1 Remapping Support

The master reconfiguration manager plays important role in generating report to master to trigger remapping. Once a PrObject detects an optimality deviation, it needs to generate a set of constraints, such as component A can't be in device B, so that the constraint resolver can apply these constraints before start the mapping computation. In this way, a PrObject may assist master to generate better deployment decision.

To achieve this goal, a set of predict API is implemented in the master reconfiguration manager. A PrClass may access the manager and construct a set of predictions, such as {componentId = 1, type='replica', operator = 'EQ', value = '2'}, to send to master to trigger remapping.

6.3.4.2 Reconfiguration Support

As shown in Figure 6.6, the flow based program is encoded into a compact binary format, which is called Darjeeling Archive (DJA), so that Darjeeling VM may dynamically load it with minumum transmission time through reprogramming. A DJA file contains component map, link table, initial value table, and binary WuClasses (if Java virtual wuclasses need to

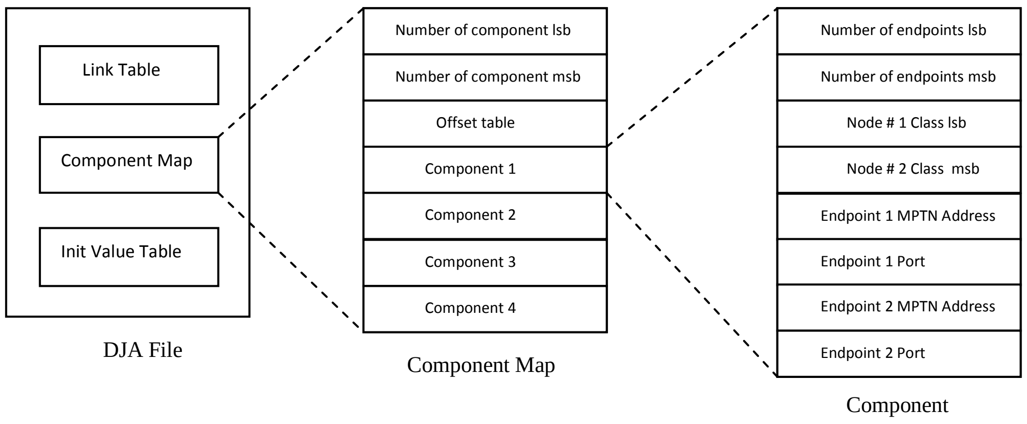


Figure 6.6: Flow Based Program Reprogramming Data

be deployed to the target device). Each component inside the component map records which physical sensors or actuaators are selected for it. A selected sensor or actuator is represented by an endpoint which includes the network address of the device and the port that the sensor or actuator is plugined. If a component has replica for the fault tolerant purpose, there will be multiple endpoints for that particular component. At the mean time, the link in DJA is represented by two component indexs in the link table.

During run-time, if a PrObject want to reconfig a small part of application, the most effective way is to change senders and receivers of existing links. Since each of them is distributedly deployed in a WuKong system, a reconfiguration algorithm needs be designed to make sure there is no abnormal things happen during transitional state, and the changing is consistent on both side. In order to maintain robust of the process, an endpoint changing mechanism [107] is designed to resolve the issue of replica replacement.

In the mechanism, locks are generated and passed to downstream mapped services to indicate upstream data is not reliable for now. The use of locks grant more global knowledge of the component endpoint changing states. Once a lock is seen in a WuDevice, services in the WuDevice will treat upstream property data change as unreliable. A lock is composed with parts: lock ID, component ID.

To facilitate PrClass reconfigure a part of application, a set of API is implemented in application reconfiguration manager. Given an application with three components C_i, C_j, C_k, service S_j is on device D_b and its backup service is on $S_{j'}$ device $D_{b'}$. A link L_{ij} is the upstream link to S_j, and another link L_{jk} is its downstream link. If we want to replace S_i with $S_{i'}$, we need to use to protocol to sequentially notify devices. To provide programming simplicity, we provide three component changing primitives:

- **Set Lock:** Send a WKPF set lock message (C_j) to target device D_a, and notify the device D_a to stop sending and receiving messages to and from all of the endpoints of the component (C_j).
- **Change Component:** Send a WKPF change component message $(C_j, S_j, S_{j'})$ to target device D_a to ask the device to update its component map item (S_j) to $(S_{j'})$
- **Release Lock:** Send a WKPF release lock message (C_j) to target device D_a to clear the lock. Thus, the messages processing can be recovered.

Algorithm 6.1 Replica Replacement Protocol

Input: The faulty device D_f and a set of in use component $F(D_f)$ whose primary endpoint is on the device

```
1: for all faulty component $C_j \in F(D_f)$ do
2:     for all component $C_i$ that $(C_i, C_j)$ is a link in link table do
3:         Send set lock to device $D_{C_i}$ for link $(C_i, C_j)$
4:         Send change component $(C_j, S_j, S_{j'})$ to $D_{C_i}$
5:     end for
6:     for all component $C_k$ that $(C_j, C_k)$ is a link in link table do
7:         Send set lock to device $D_{C_k}$ for link $(C_j, C_k)$
8:         Send change component $(C_j, S_j, S_{j'})$ to $D_{C_k}$
9:     end for
10:    for all component $C_i$ that $(C_i, C_j)$ is a link in link table do
11:        Send release lock to device $D_{C_i}$ for link $(C_i, C_j)$
12:    end for
13:    for all component $C_k$ that $(C_j, C_k)$ is a link in link table do
14:        Send release lock to device $D_{C_k}$ for link $(C_j, C_k)$
15:    end for
16: end for
```

In the protocol, the D_{C_i} denotes the device that currently hosts the primary endpoint of component C_i. With these three primitives, the fault recovery algorithm is implemented in progression server to handle with failure of devices. In section 6.4.1, we will discuss the fault tolerant policy as a use case in detail.

6.4 Progressive System Case Study

In this section, we will elaborate how to use progression framework to implement proactive autonomic management of application and systems. Section 6.4.1 discusses how to use reconfiguration API to implement fault tolerant policy. After that, the implementation of occupancy aware policy with the remapping support is discussed in section 6.4.2.

6.4.1 Reconfiguration for Fault Tolerant

To achieve application reliability in WuKong middle-ware, fault tolerant policy is an important progressive capability to support. The fault tolerant policy aims to support dynamic reconfiguration of applications at run-time, in response to device failure through resilient substitution that can be applied to any component or sub DAG of the FBP. To eliminate the single failure of master and release master from heavily monitoring overhead of a large number of applications, the fault tolerant policy PrClass is designed for fault recovery of an application. If a fault tolerant policy is chosen, the component is added by master before deployment.

In the fault tolerant mapping policy, a component of an application may have 1 - 2 replica, which is specificed by user through Master UI. During mapping, master will select a wuobject for each replica. Each replica of a component is chosen on different device for the purpose of fault tolerant. After remote programming, every component will initially run on the primary wuobject, and other wuobjects chosen for replicas will be in standby status.

During run-time, the fault tolerant policy PrObject in progression server will periodically send heartbeat request to the each devices that host at least one endpoint of a replicated component. If a device can't reply to the heartbeat request for three times, PrObject will treat it as faulty device. Then, the PrObject is responsible for initializing a component change process to replace the failed primary instance with the replica and resume the execution.

6.4.2 User Occupancy Aware Remapping

In smart home environment, an intelligent application needs to be smart enough to function differently for different user context. For example, the nest device may detect whether user is at home or not. If no user exists or user is going to leave, it will close the air condition. It may also find users' existence pattern from historical data, and predict the coming time of user and restart air condition. In this section, we provide an implementation of the user occupancy aware application policy. So that application may easily change mode when user switch between present and absent.

Algorithm 6.2 Occupancy Prediction Algorithm

```
Input: $V_c$ and a set of vector $V_i$ as $S(V_i)$
Input: current time slot k
Output: probability of occupancy in time slot $length(V_c) + 1$

1: Init a maximum heap H, in which items are sorted by distance
2: for all vector $V_i$ in the set $S(V_i)$ do
3:     calculate hamming distance $d_i$ between $V_c$ and $V_i$ until slot i
4:     put the pair $(i, d_i)$ into heap H
5: end for
6: pop top K items in the H
7: find and put such K vectors by its index into a set $S_{top}$
8: calculate the mean M of vector $V_k(i + 1)$, $V_k \in S_{top}$
9: return M
```

To study how we may switch mode of applications through prediction from historical users's pattern, we use the occupancy prediction algorithm defined by the research of PreHeat [84] in Microsoft. The sensing of occupancy is achieved by adding motion sensors in rooms. All motion sensors in the system keep on reporting its value to progression server, so that the PrClass need to use a system buffer to hold it. Given historical occupancy vectors and occupancy vector until time slot k, the detailed algorithm to predict occupancy in time slot k + 1 is presented below.

In the Algorithm 6.2, space occupancy is represented by a binary vector for each day, where each element represents occupancy in a 15 minute interval. The vector element is 1 if there is any occupancy during a partial during the interval or 0 otherwise. As a day progresses, system maintain a partial vector from midnight up to the current time. To predict future occupancy, we use the partial occupancy vector to find similar days in the past. Specifically, we compute the Hamming distance between the current partial day and the corresponding parts of all the past occupancy vectors. (The Hamming distance simply counts the number of unequal corresponding binary vector elements.) We then pick the K nearest past days for making the prediction. Based on initial experiments, we found K=5 proved to be a good choice for high prediction accuracy. The predicted occupancy probability for a future time is simply the mean of the corresponding occupancy values in the K nearest past days.

6.5 System Performance Study

In this section, we study performance of progression framework in terms of remapping latency, reconfiguration latency, model update latency and reprogramming efficiency in a management zone.

6.5.1 System Setup

We setup a management zone for performance study in our lab. As shown in Figure 6.7, the management zone contains a progression server, two reprogramming delegator and three gateways. Each of them are standone software component on separate devices. WuKong Master, XMPP and MongoDB are run in a Mini-Mac.

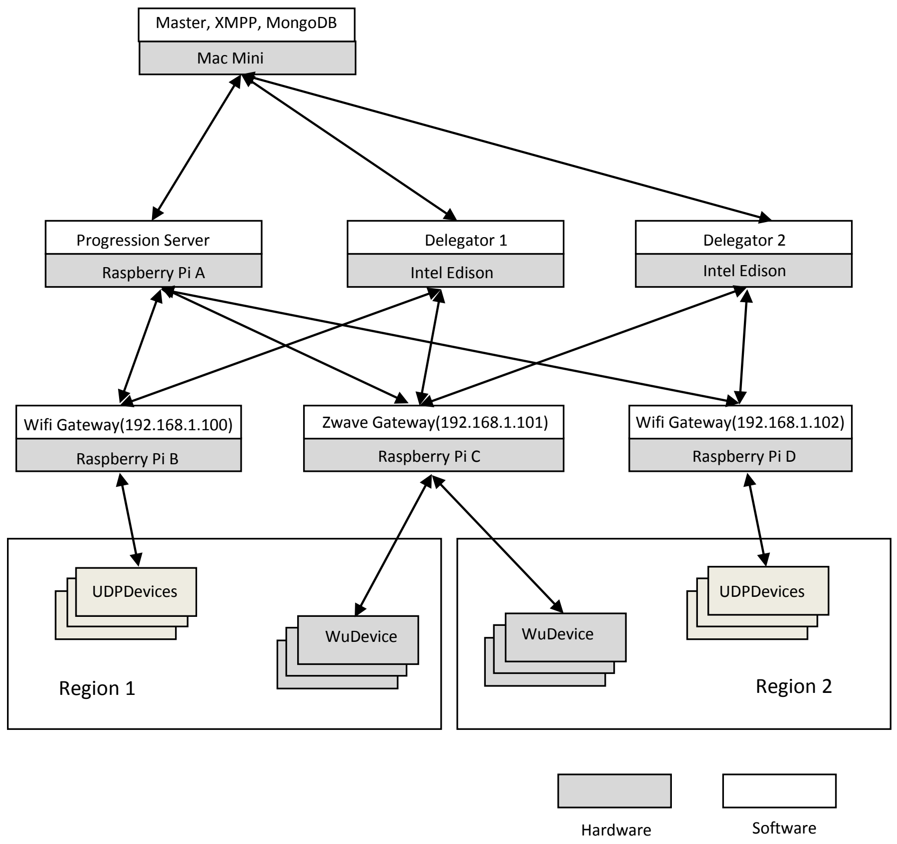


Figure 6.7: System Setup for Performance Study

Within the management zone, there are two regions connected by three gateways. Two separate delegaters on intel edison are responsible for reprograming these two regions. Each of gateways are run on a Rapsberry Pi. We include two types of devices in the study, which are UDPDevices connected with UDPGateway and WuDevice connected with ZwaveGateway.

For those UDPDevices, they are actually python programs that implement the WKPF, so that they may provide service for computation. Each of them are collocated with the gateway its belongs to on Raspberry Pi. Each of Wudevice is actually a physical device that runs WuKong Darjeeling to control sensors and actuators. Under each UDPGateway, there are 8 UDPDevices. The ZwaveGateway connects with 16 wudevices. Each Wudevice has ATMega 2560 micro-controller, 32KB EPROM, 3 digital I/O and 3 analog I/O, and nested zwave communication module. Our Wukong darjeeling JVM is ported onto each Wudevice. The Raspberry Pi 2 has 900MHz quad-core ARM Cortex-A7 CPU and 1 GB RAM, and progression server is deployed on it as a streaming hub.

6.5.2 Remapping Latency

In this round of study, we deploy FBPs with different size with user aware policy PrClass in the first region. If a status (absent/occupancy) is predicted different with current status, the remapping request will be sent to master. Since remapping is to trigger reprogramming the same FBP with different deployment decision, their efficiency is impacted by the intensity of network traffic. Thus, we use a backgroud FBP to generate network traffice from a UDPDevice to a ZwaveDevice. The UDPDevice generates network traffic in 60, 80, 120, 240 packets per minute.

Figure 6.8 shows the remapping latency linearly grows with the size of FBP under normal network traffic (120 packets per minute). But under high traffic (240 packet per minute), the latency increased obviously because of the network congestion on the Zwave Dongle. Althrough high traffice brings extra delay in remapping, we may achieve a remapping of a FBP with 16 components within 2 minites. The latency is acceptable for context update of IoT systems.

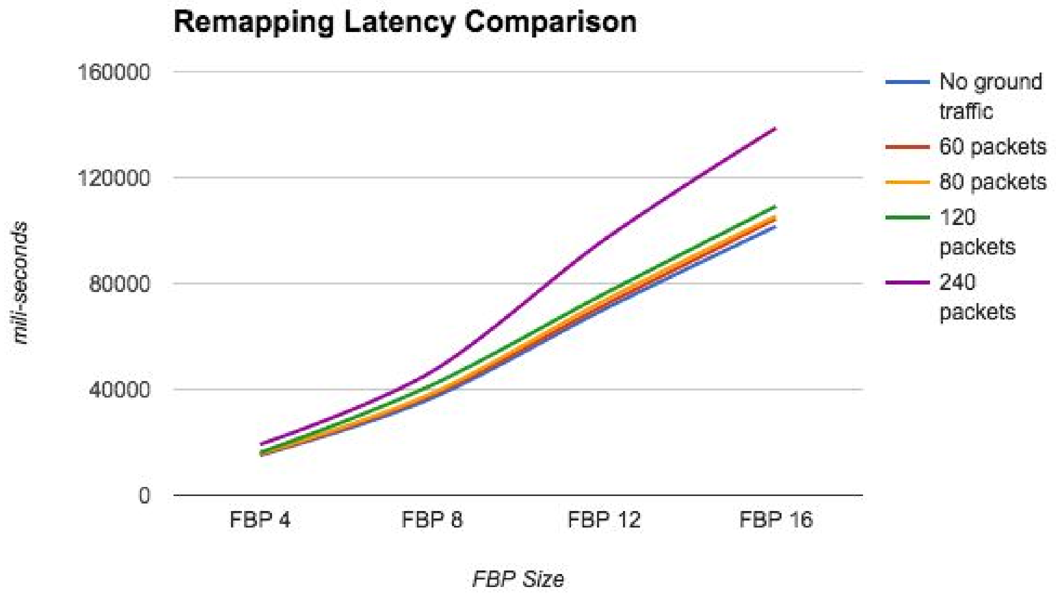


Figure 6.8: Remapping Latency

6.5.3 Reconfiguration Latency

To study the efficiency of reconfiguration, the fault tolerant scenarios is used for evaluate how long it takes to recover a faulty component. As shown in 6.4.1, the recover time is determined by three factors: the fault detection time (how long to treat heartbeat as timeout), the degree of the faulty component in FBP, and the network traffic. Since the timeout parameter depends on the setting of algorithm for different application scenarios, we mainly study how the degree of component and network traffic will impact the recover time. As what we did for remapping latency study, latency for network traffic in 60, 80, 120, 240 packets per minute are compared.

In Figure 6.9, re component with different degree (2, 4, 6, 8) are compared. For each setting, the latency of replacement for each type of setting is the average of 10 times physical latency minus the fault detection time parameter (3 * 2 seconds). The replica replacement protocol includes three steps: lock, change map, release lock. Since request messages of each type are parallel send to each device, the total time is actually three round trip time in WuKong MPTN. When the traffic is relative low, the average round trip time is about 200ms. Thus,

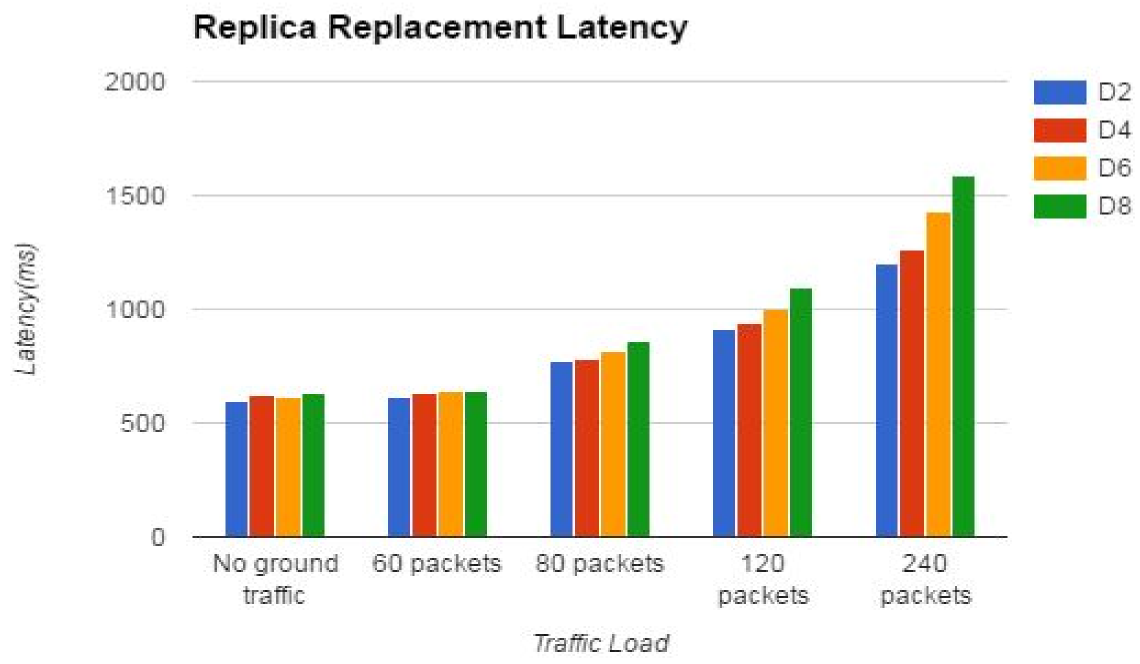


Figure 6.9: Component Replacement Latency

the latency is about 200ms to 800ms in the network traffic of (0 to 80 packets per minutes). As traffic grows to 120 packets per minute, a tranmission lag in data forarding queue of zwave gateway is observed. It is because of the limited bandwidth of Zwave network. Therefore, the latency is also most doubled in 120 packets per minute setting, and tripled in 240 packets per minute. But even in high network traffic, the component replacement procotol may still be executed within 1.5 seconds, which is acceptable in most of home and office application scenarios.

6.5.4 Model Update Latency

In this group of study, I measure the edge model update latency when the size of model content grows. The user preference control PrClass is used to perform the study. This PrClass listens to a topic called preference from XMPP. The size of json file which is records the preference table is updated in each round of study, to see how long can it be applied to PrClass for control. Since XMPP is also used as a storage for checkpointing of Edge Classes. To simulate these kind of system overhead, an extra XMPP client is used to periodically

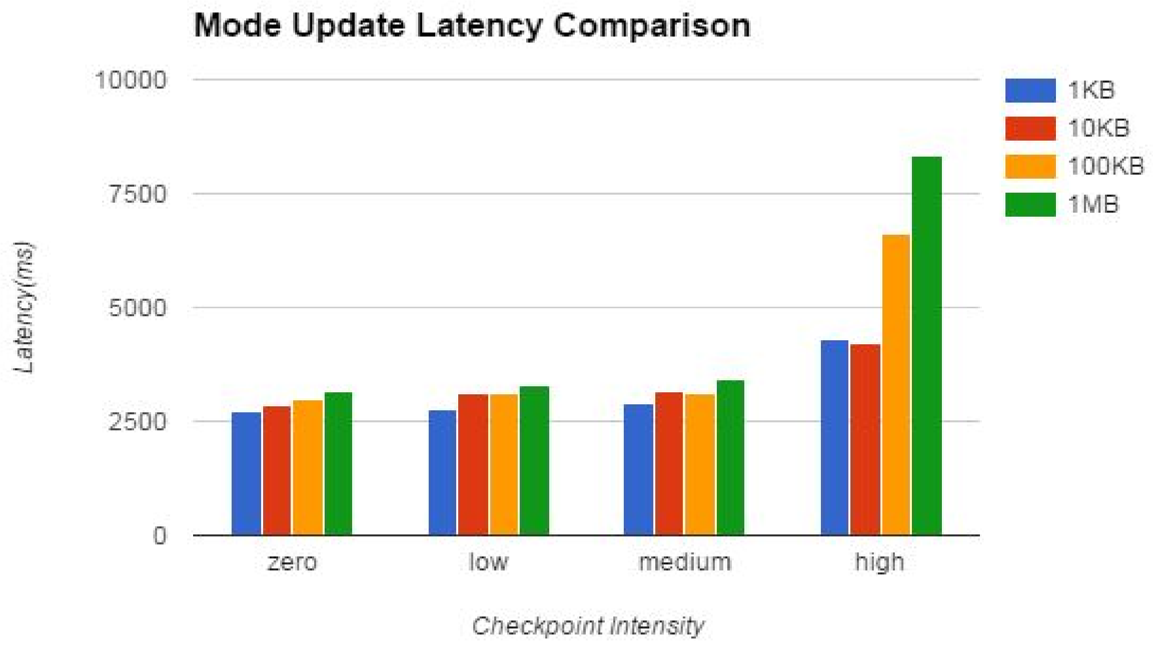


Figure 6.10: Model Update Latency

Figure 6.10 shows that the model update is pretty fast (less than 3 seconds) when the model size is less than 100KB. But when the size grows to 1MB, the latency grows to 8 seconds. It is because of 1MB needs multiple round of communication between edge server and XMPP to delivery big model. Since parameters size of most of machie learning alogorithm that can applied in edge directly is less than 1MB, we believe it is an acceptable latency for edge application.

6.5.5 Scalable Reprogramming Performance

In Chapter 4, we have introdiced the scalable deployment algorithms static mapping algorithm, uniform mapping algorithm, and optimal mapping algorithm. We also study how our optimal algorithm outperform other algorithms in a real system. In this study, we need to reprogramming both two regions. The zwave gateway connects Wudevices belongs to both two regions, so that it is a congest gateway and these two regions form a congestion zone.

In this study, three types of reprogramming strategies, sequential reprogramming, distributed static reprogramming, and distributed optimal reprogramming are compared. In sequential reprogramming strategy, the master sequentially to reprogram both two regions, so the reprogramming time is the sum of deploy time on region 1 and region 2. In the distributed static reprogramming, region 1 and region 2 are parallel reprogramed by delegator 1 and delegator 2, and the static algorithm is applied in the mapping decision. In distributed optimal reprogramming, we not only use parallel reprogramming, but also balance the traffic of each of gateway with the optimal mapping algorithm proposed in Section 4.2.

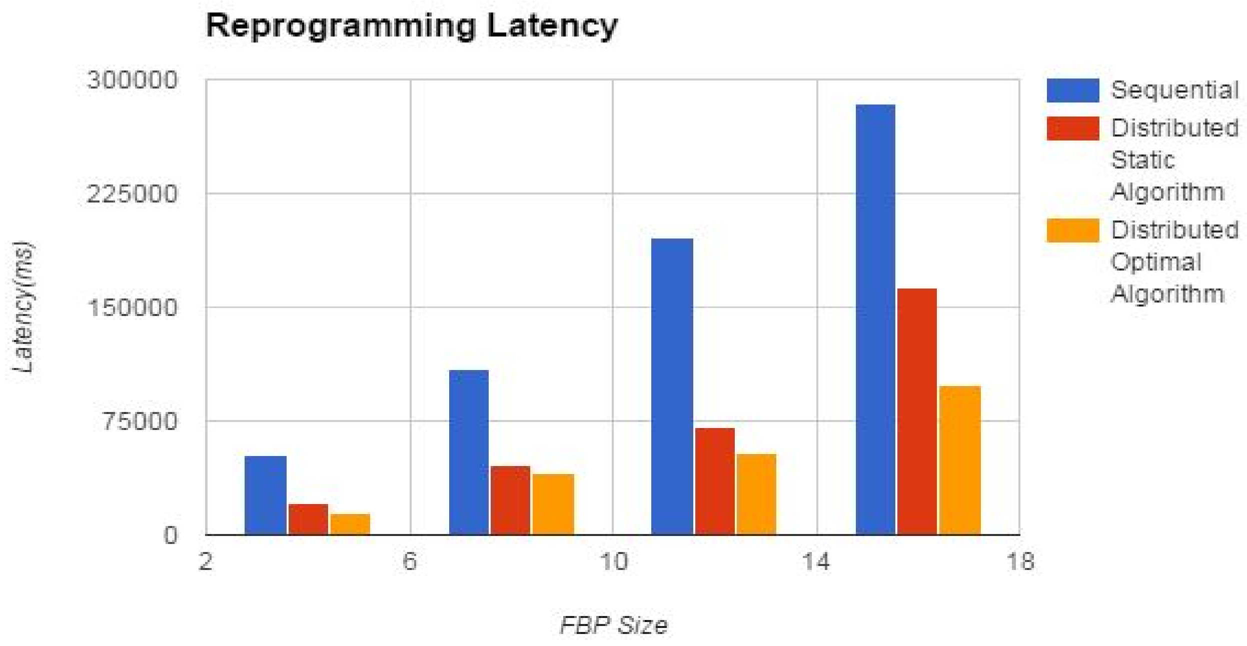


Figure 6.11: Reprogramming Algorithm Comparison

In the study, we perform 3 rounds of reprogramming for each size of FBP (4, 8, 12, 16) with each type of strategies. As shown in Figure 6.11, distributed static reprogramming algorithm may reduce the latency to less than 1/n (n is the number of region in a congestion zone), due to the gain of paralle reprogramming. Distributed optimal algorithm may further reduce 15% to 45% of reprogramming time, which is better than 15% to 30% in simulation. The reason behind it is that there are more extra delay in each round of reprogramming packet passing in the congested gateway. Our optimal algorithm reduce the traffic in the congested gateway, so also reduce the packet transmission time. We can conclude that the optimal algorithm may have even more improvement in a bigger congestion zone in which more devices are involved.

6.6 Summary of Progressive System

In this chapter, we investegate the emergent requirment of proactive management of IoT systems on the edges of network. In our research, the progression model is identified as self-learning, policy-driven and context aware. This chapter presents a system architecture, progression framework, that is built for self learning based self management of IoT applications during runtime. This design release the master from heavy workload of runtime management of applications deployed on the system.

Beside supporting streaming analytics and simple programming paradigm, progression server builds efficient addons for system monitoring and programming primitives for remapping, reconfiguring systems. Our evaluation has demonstrated that progression framework provides an efficient, yet open autonomic management framework for IoTs in edge. With our self-learning support and reconfiguration support in the framework, diversified policies for IoT application may be designed and managed within the progression framework during runtime.

Chapter 7

Conclusions and Future Work

The large scale and heterogeous attributes of IoT system bring issues, including energy saving, efficient and low latency deployment and management. At the mean time, the large amount of data generated by devices generate extra load on current centralized big data system, and calls for edge computing support for building intelligence near the data.

In this dissertation, I studied the scalability of service oriented IoT system, and proposed an edge and progression framework for building edge instelligence and predictive maintained applications in service oriented IoT systems. The contribution of this dissertation is as follow:

- Energy aware mapping algorithms are designed to save communication cost for IoT application. If an application to be deployed in multi-hop network, the optimal mapping can be efficiently resolved by quadratic programming algorithm. If it is in a single-hop network, the optimization complexity can be further reduced by applying greedy algorithm of MWIS in collocation graph.
- Large scale low latency application deployment is usually needed for emergency man-

agement in public administration. The mapping of thousands of copies of single application is studied. In the study, target area is divided into zone that doesn't have overlapped network congestion. Within each zone, mapping algorithm that considering both repogramming latency and runtime deadline is devised.

- To support edge intelligence, the edge framework is designed for building online learning on streaming data. Within the edge server, native data structure and time series operators provides programming simpilicity for IoT application developer. Moreover, the edge class level fault tolerant is achieved through check pointing and recovery.
- As the intelligent management solution, the progression framework is built for online optimality of mapping decision. It is built as a plug-and-play streaming server that is capable of cooperating with devices and master to achieve self-management and self-configuration.

The increasing number of large scale IoT application makes demanding requirements on scalable management solution in middleware layer. We believe that our edge and progression framework design is a practical and efficient solution in meeting this challenge

Bibliography

[1] Cloud index forcast. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf.

[2] Dasada. http://www.rl.af.mil/tech/programs/dasada/program-overview.html.

[3] Gartner market research. http://www.gartner.com/newsroom/id/3165317.

[4] Nano vm. http://www.harbaum.org/till/nanovm/index.shtml/.

[5] Samza. http://samza.apache.org//.

[6] Sas. http://www.darpa.mil/ato/programs/suosas.htm.

[7] Wsu casas dataset. http://ailab.wsu.edu/casas/datasets/.

[8] Wukong darjeeling. https://github.com/wukong-m2m/wukong-darjeeling.

[9] G. T. A, G. S. B, S. T. A, and P. N. B. A self-managing infrastructure for ad-hoc situation determination, 2006.

[10] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

[11] S. Agarwala, Y. Chen, D. Milojicic, and K. Schwan. Qmon: Qos- and utility-aware monitoring in enterprise systems. In *2006 IEEE International Conference on Autonomic Computing*, pages 124–133, June 2006.

[12] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in meteor-s. In *Proceedings of the 2004 IEEE International Conference on Services Computing*, SCC '04, pages 23–30, Washington, DC, USA, 2004. IEEE Computer Society.

[13] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 881–890, New York, NY, USA, 2009. ACM.

[14] R. Angarita. Responsible objects: Towards self-healing internet of things applications. In *Autonomic Computing (ICAC), 2015 IEEE International Conference on*, pages 307–312, July 2015.

[15] F. Aslam, C. Schindelhauer, G. Ernst, D. Spyra, J. Meyer, and M. Zalloom. Introducing takatuka: A java virtualmachine for motes. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 399–400, New York, NY, USA, 2008. ACM.

[16] A. P. Athreya, B. DeBruhl, and P. Tague. Designing for self-configuration and self-adaptation in the internet of things. In *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*, pages 585–592, Oct 2013.

[17] F. S. Bao and C. Liu, Xin andZhang. Pyeeg: An open source python module for eeg/meg feature extraction. *Computational Intelligence and Neuroscience*, (7), 2011.

[18] A. Billionnet and A. Faye. A lower bound for a constrained quadratic 0/1 minimization problem. *Discrete Applied Mathematics*, 74(2):135 – 146, 1997.

[19] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple object access protocol (soap) 1.1, 2000.

[20] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, Apr. 1998.

[21] T. Broderick, N. Boyd, A. Wibisono, A. C. Wilson, and M. Jordan. Streaming variational bayes. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1727–1735. 2013.

[22] N. Brouwers, K. Langendoen, and P. Corke. Darjeeling, a feature-rich vm for the resource poor. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 169–182, New York, NY, USA, 2009. ACM.

[23] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava. Adaptation in web service composition and execution. In *Proc. of IEEE International Conference on Web Services (ICWS)*, 2006.

[24] C.-Y. Chang and H.-R. Chang. Energy-aware node placement, topology control and mac scheduling for wireless sensor networks. *Comput. Netw.*, 52(11):2189–2204, Aug. 2008.

[25] D. J. Cook, N. C. Krishnan, and P. Rashidi. Activity discovery and activity recognition: A new partnership. *IEEE Transactions on Cybernetics*, 43(3):820–828, June 2013.

[26] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[27] T. Dumitras and P. Narasimhan. Fault-tolerant middleware and the magical 1%. In *Middleware*, pages 431–441, 2005.

[28] V. L. Erickson, M. . Carreira-Perpin, and A. E. Cerpa. Observe: Occupancy-based system for efficient reduction of hvac energy. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 258–269, April 2011.

[29] M. E. Femal and V. W. Freeh. Boosting data center performance through non-uniform power allocation. In *Second International Conference on Autonomic Computing (ICAC'05)*, pages 250–261, June 2005.

[30] R. T. Fielding. *Architectural styles and the design of network-based software architectures.* PhD thesis, University of California, 2000.

[31] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.

[32] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Syst. J.*, 42(1):5–18, Jan. 2003.

[33] X. Gu and K. Nahrstedt. Distributed multimedia service composition with statistical QoS assurances. *IEEE Transactions on Multimedia*, 8(1):141–151, 2006.

[34] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *Services Computing, IEEE Transactions on*, 3(3):223–235, July 2010.

[35] L. Gurgen, O. Gunalp, Y. Benazzouz, and M. Gallissot. Self-aware cyber-physical systems and applications in smart buildings and cities. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1149–1154, March 2013.

[36] M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 439–448, New York, NY, USA, 1994. ACM.

[37] J. Hastad. Clique is hard to approximate within n1- epsiv;. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 627–636, Oct 1996.

[38] T. Haveliwala. Efficient computation of pagerank. Technical Report 1999-31, Stanford InfoLab, 1999.

[39] T. H. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 517–526, New York, NY, USA, 2002. ACM.

[40] Q. He, J. Yan, H. Jin, and Y. Yang. Adaptation of web service composition based on workflow patterns. In *Proc. of International Conference on Service-Oriented Computing (ICSOC)*. Springer, 2008.

[41] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *Wireless Communications, IEEE Transactions on*, 1(4):660–670, 2002.

[42] P. Horn. Autonomic computing: Ibm's perspective on the state of information technology. `http://researchweb.watson.ibm.com/autonomic`, 2001.

[43] Z. Huang, W. Jiang, S. Hu, and Z. Liu. Effective pruning algorithm for QoS-aware service composition. In *Commerce and Enterprise Computing, 2009. CEC '09. IEEE Conference on*, pages 519–522, 2009.

[44] Z. Huang, K.-J. Lin, and A. Han. An Energy Sentient Methodology for Sensor Mapping and Selection in IoT Systems. *2014 IEEE International Symposium on Industrial Eletronics*, September 2014.

[45] Z. Huang, K.-J. Lin, C. Li, and S. Zhou. Communication energy aware sensor selection in iot systems. In *2014 IEEE and Internet of Things (iThings/CPSCom)*, September 2014.

[46] Z. Huang, K.-J. Lin, S.-Y. Yu, and J. Y.-j. Hsu. Building energy efficient internet of things by co-locating services to minimize communication. In *Proceedings of the 6th International Conference on Management of Emergent Digital EcoSystems*, MEDES '14, pages 18:101–18:108, New York, NY, USA, 2014. ACM.

[47] Z. Huang, K.-J. Lin, S.-Y. Yu, and J. Y.-j. Hsu. Building energy efficient internet of things by co-locating services to minimize communication. In *Proceedings of the 6th International Conference on Management of Emergent Digital EcoSystems*, MEDES '14, pages 18:101–18:108, New York, NY, USA, 2014. ACM.

[48] Z. Huang, B. L. Tsai, J. J. Chou, C. Y. Chen, C. H. Chen, C. C. Chuang, K. J. Lin, and C. S. Shih. Context and user behavior aware intelligent home control using wukong middleware. In *Consumer Electronics - Taiwan (ICCE-TW), 2015 IEEE International Conference on*, pages 302–303, June 2015.

[49] M. C. Huebscher and J. A. McCann. Adaptive middleware for context-aware applications in smart-homes. In *Proceedings of the 2Nd Workshop on Middleware for Pervasive and Ad-hoc Computing*, MPAC '04, pages 111–116, New York, NY, USA, 2004. ACM.

[50] P. Jayachandran and T. Abdelzaher. Delay composition in preemptive and non-preemptive real-time pipelines. *Real-Time Syst.*, 40(3):290–320, 2008.

[51] W. Jiang, C. Zhang, Z. Huang, M. Chen, S. Hu, and Z. Liu. QSynth: A tool for QoS-aware automatic service composition. In *IEEE International Conference on Web Services (ICWS)*, pages 42–49, 2010.

[52] N. Kandasamy, S. Abdelwahed, and J. P. Hayes. Self-optimization in computer systems via on-line control: application to power management. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 54–61, May 2004.

[53] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[54] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan. 2003.

[55] B. Khargharia, S. Hariri, and M. Yousif. *Autonomic power and performance management for computing systems*, volume 2006, pages 145–154. 2006.

[56] S. S. Kulkarni and L. Wang. Mnp: Multihop network reprogramming service for sensor networks. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 7–16, June 2005.

[57] P. Levis and D. Culler. MatÉ: A tiny virtual machine for sensor networks. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, pages 85–95, New York, NY, USA, 2002. ACM.

[58] K. Lin, M. Panahi, Y. Zhang, J. Zhang, and S. Chang. Building accountability middleware to support dependable SOA. *IEEE Internet Computing*, 13:16–25, 2009.

[59] K. Lin, J. Zhang, Y. Zhai, and B. Xu. The design and implementation of service process reconfiguration with end-to-end qos constraints in soa. *Service Oriented Computing and Applications*, 4:157–168, 2010.

[60] K. J. Lin, M. Panahi, Y. Zhang, J. Zhang, and S.-H. Chang. Building accountability middleware to support dependable SOA. *IEEE Internet Computing*, 13(2):16–25, 2009.

[61] K.-J. Lin, N. Reijers, Y.-C. Wang, C.-S. Shih, and J. Y. Hsu. Building Smart M2M Applications Using the WuKong Profile Framework. *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 1175–1180, Aug. 2013.

[62] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, Jan. 2003.

[63] G. Lu, N. Sadagopan, B. Krishnamachari, and A. Goel. Delay efficient sleep scheduling in wireless sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2470–2481 vol. 4, 2005.

[64] K. Mechitov and G. Agha. Software service and application engineering. chapter An Architecture for Dynamic Service-oriented Computing in Networked Embedded Systems, pages 147–164. Springer-Verlag, Berlin, Heidelberg, 2012.

[65] D. A. Menascé. Qos issues in web services. *IEEE Internet Computing*, 6(6):72–75, Nov. 2002.

[66] Microsoft Corporation. *Distributed Component Object Model Protocol (DCOM)*, 1.0 edition, Jan. 1998.

[67] M. Milenkovic and O. Amft. An opportunistic activity-sensing approach to save energy in office buildings. In *Proceedings of the Fourth International Conference on Future Energy Systems*, e-Energy '13, pages 247–258, New York, NY, USA, 2013. ACM.

[68] H. Moussa, T. Gao, I.-L. Yen, F. Bastani, and J.-J. Jeng. Toward effective service composition for real-time soa-based systems. *Service Oriented Computing and Applications*, 4:17–31, 2010.

[69] R. Murch. *Autonomic Computing (On Demand Series)*. IBM Press, 2004.

[70] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Proceedings of the Second International Conference on Automatic Computing*, ICAC '05, pages 229–240, Washington, DC, USA, 2005. IEEE Computer Society.

[71] V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and energy efficient data dissemination service for extreme scale wireless networks of embedded devices. *IEEE Transactions on Mobile Computing*, 6(7):777–789, July 2007.

[72] M. Nakamura and L. D. Bousquet. Constructing execution and life-cycle models for smart city services with self-aware iot. In *Autonomic Computing (ICAC), 2015 IEEE International Conference on*, pages 289–294, July 2015.

[73] S. Olariu and I. Stojmenovic. Design guidelines for maximizing lifetime and avoiding energy holes in sensor networks with uniform distribution and uniform reporting. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, 2006.

[74] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1099–1110, New York, NY, USA, 2008. ACM.

[75] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '99, pages 15–15, Berkeley, CA, USA, 1999. USENIX Association.

[76] M. Panahi, W. Nie, and K.-J. Lin. The design and implementation of service reservations in real-time soa. *E-Business Engineering, IEEE International Conference on*, 0:129–136, 2009.

[77] M. Panahi, W. Nie, and K.-J. Lin. The design of middleware support for real-time soa. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011 14th IEEE International Symposium on*, pages 117 –124, march 2011.

[78] S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for web service composition. In *11th World Wide Web Conference, Honolulu, Hawaii*, May 2002.

[79] P. Pradhan, V. Baghel, G. Panda, and M. Bernard. Energy efficient layout for a wireless sensor network using multi-objective particle swarm optimization. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 65–70, 2009.

[80] N. Reijers, K.-J. Lin, Y.-C. Wang, C.-S. Shih, and J. Y. Hsu. Design of an intelligent middleware for flexible sensor configuration in M2M systems. In *SENSORNETS*, pages 41–46, 2013.

[81] N. Reijers, Y.-C. Wang, C.-S. Shih, J. Y. jen Hsu, and K.-J. Lin. Building intelligent middleware for large scale CPS systems. In *IEEE Conference on Service-Oriented Computing and Applications*, pages 1–4, 2011.

[82] S. Sakai, M. Togasaki, and K. Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, 126(2 - 3):313 – 322, 2003.

[83] J. Scott, A. Bernheim Brush, J. Krumm, B. Meyers, M. Hazas, S. Hodges, and N. Villar. Preheat: Controlling home heating using occupancy prediction. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 281–290, New York, NY, USA, 2011. ACM.

[84] J. Scott, A. Bernheim Brush, J. Krumm, B. Meyers, M. Hazas, S. Hodges, and N. Villar. Preheat: Controlling home heating using occupancy prediction. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 281–290, New York, NY, USA, 2011. ACM.

[85] C.-S. Shih and G.-F. Wu. Distributed meta-routing over heterogeneous networks for m2m/iot systems. In *Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems*, RACS, pages 443–450, New York, NY, USA, 2015. ACM.

[86] M. Stolikj, P. J. L. Cuijpers, and J. J. Lukkien. Efficient reprogramming of wireless sensor networks using incremental updates. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, pages 584–589, March 2013.

[87] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas. Service oriented middleware for the internet of things: A perspective. In *Proceedings of the 4th European Conference on Towards a Service-based Internet*, ServiceWave'11, pages 220–229, Berlin, Heidelberg, 2011. Springer-Verlag.

[88] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629, Aug. 2009.

[89] W. Torres-Pomales. *Software Fault Tolerance: A Tutorial.* Digital Library Network for Engineering and Technology, Oct 2000.

[90] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM.

[91] J. Wang, D. Li, G. Xing, and H. Du. Cross-layer sleep scheduling design in service-oriented wireless sensor networks. *IEEE Transactions on Mobile Computing*, 9(11):1622–1633, 2010.

[92] Q. Wang, M. Hempstead, and W. Yang. A realistic power consumption model for wireless sensor network devices. In *3rd IEEE Conference on Sensor and Ad Hoc Communications and Networks, SECON '06.*, volume 1, pages 286–295, 2006.

[93] Q. Wang, Y. Zhu, and L. Cheng. Reprogramming wireless sensor networks: challenges and approaches. *IEEE Network*, 20(3):48–55, May 2006.

[94] Y.-C. Wang, W.-C. Peng, and Y.-C. Tseng. Energy-balanced dispatch of mobile sensors in a hybrid wireless sensor network. *Parallel and Distributed Systems, IEEE Transactions on*, 21(12):1836–1850, 2010.

[95] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1567–1576 vol.3, 2002.

[96] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 12(3):493–506, June 2004.

[97] T. Yu and K. Lin. Adaptive algorithms for finding replacement services in autonomic distributed business processes. In *Proc. of the 7th International Symposium on Autonomous Decentralized Systems (ISADS2005)*, pages 427–434, 2005.

[98] T. Yu and K. Lin. Service selection algorithms for web services with end-to-end QoS constraints. *Information Systems and E-Business Management*, 3(2):103–126, 2005.

[99] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end QoS constraints. *ACM Trans. Web*, 1(1), May 2007.

[100] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[101] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, HotCloud'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[102] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web*, pages 411–421. ACM Press New York, NY, USA, 2003.

[103] Y. Zhai, J. Zhang, and K. J. Lin. Soa middleware support for service process reconfiguration with end-to-end qos constraints. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 815–822. IEEE, 2009.

[104] Y. Zhang, K. Lin, and J. Hsu. Accountability monitoring and reasoning in service-oriented architectures. *Springer Journal on Service Oriented Computing and Applications*, 1(1):35–50, 2007.

[105] Y. Zhang, K. J. Lin, and J. Y. Hsu. Accountability monitoring and reasoning in service-oriented architectures. *Journal of Service-Oriented Computing and Applications (SOCA)*, 1(1), 2007.

[106] Y. Zhang, M. Panahi, and K. J. Lin. Service process composition with QoS and monitoring agent cost parameters. *IEEE Joint Conf. on E-Commerce Technology (CEC'08) and Enterprise Computing, (EEE '08)*, July 2008.

[107] S. Zhou, K. J. Lin, J. Nay, C.-C. Chuangz, and C.-S. Shih. Supporting service adaptation in fault tolerant internet of things. In *IEEE Conference on Service-Oriented Computing and Applications*, 2015.