Newcastle University

School of Computing Science

Simplifying Internet of Things (IoT) Data Processing Workflow Composition and Orchestration in Edge and Cloud Datacenters

Yinhao Li

Submitted for the degree of Doctor of Philosophy in the School of Computing Science, Newcastle University

December 2020

Abstract

Internet of Things (IoT) allows the creation of virtually infinite connections into a global array of distributed intelligence. Identifying a suitable configuration of devices, software and infrastructures in the context of user requirements are fundamental to the success of delivering IoT applications. However, the design, development, and deployment of IoT applications are complex and complicated due to various unwarranted challenges. For instance, addressing the IoT application users' subjective and objective opinions with IoT workflow instances remains a challenge for the design of a more holistic approach. Moreover, the complexity of IoT applications increased exponentially due to the heterogeneous nature of the Edge/Cloud services, utilised to lower latency in data transformation and increase reusability.

To address the composition and orchestration of IoT applications in the cloud and edge environments, this thesis presents IoT-CANE (Context Aware Recommendation System) as a high-level unified IoT resource configuration recommendation system which embodies a unified conceptual model capturing configuration, constraint and infrastructure features of Edge/Cloud together with IoT devices. Second, I present an IoT workflow composition system (IoTWC) to allow IoT users to pipeline their workflows with proposed IoT workflow activity abstract patterns. IoTWC leverages the analytic hierarchy process (AHP) to compose the multi-level IoT workflow that satisfies the requirements of any IoT application. Besides, the users are befitted with recommended IoT workflow configurations using an AHP based multi-level composition framework. The proposed IoTWC is validated on a user case study to evaluate the coverage of IoT workflow activity abstract patterns and a real-world scenario for smart buildings. Last, I propose a fault-tolerant automation deployment IoT framework which captures the IoT workflow plan from IoTWC to deploy in multi-cloud edge environment with a fault-tolerance mechanism. The efficiency and effectiveness of the proposed fault-tolerant system are evaluated in a real-time water flooding data monitoring and management application.

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration unless otherwise stated. It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the Newcastle University or any other University or similar institution.

Yinhao Li,

December 2020

Publications

Published

1. **Y. Li**, A. Alqahtani, E. Solaiman, C. Perera, P. P. Jayaraman, R. Buyya, G. Morgan, and R. Ranjan, "IoT-CANE: A unified knowledge management system for data-centric Internet of Things application systems." Journal of Parallel and Distributed Computing 131 (2019): 161-172.

2. **Y. Li**, D.N.Jha, G.S.Aujla, G.Morgan, A.Y.Zomaya and R.Ranjan, 2020, April. "IoTWC: Analytic Hierarchy Process Based Internet of Things Workflow Composition System." In 2020 IEEE International Conference on Cloud Engineering (IC2E), pp. 1-10. IEEE, 2020.

3. D. N. Jha, Z. Wen, **Y. Li**, M. Nee, M. Koutny and R. Ranjan, 2020, January. "A Cost-Efficient Multi-cloud Orchestrator for Benchmarking Containerized Web-Applications." In International Conference on Web Information Systems Engineering, pp. 407-423. Springer, Cham, 2020.

4. A. Alqahtani, **Y. Li**, P. Patel, E. Solaiman and R. Ranjan, 2018, July. "End-to-end service level agreement specification for iot applications." In 2018 International Conference on High Performance Computing and Simulation (HPCS), pp. 926-935. IEEE, 2018.

5. K. Alwasel, **Y. Li**, P. P. Jayaraman, S. Garg, R. N. Calheiros, and R. Ranjan, 2017. "Programming sdn-native big data applications: Research gap analysis." IEEE Cloud Computing 4, no. 5 (2017): 62-71.

6. B. Qian, J. Su, Z. Wen, D. N. Jha, **Y. Li**, Y. Guan, D. Puthal, P. James, R. Yang, A. Y. Zomaya and O. Rana, "Orchestrating the development lifecycle of machine learning-based iot applications: A taxonomy and survey." ACM Computing Surveys (CSUR) 53, no. 4 (2020): 1-47.

7. D. N. Jha, **Y. Li**, P. P. Jayaraman, S. Garg, P. Watson, and R. Ranjan, "Challenges in Deployment and Configuration Management in Cyber Physical System." In Handbook of Integration of Cloud Computing, Cyber Physical Systems and Internet of Things, pp. 215-235. Springer, Cham, 2020.

8. K. Alwasel, A. Noor, **Y. Li**, E. Solaiman, S. K. Garg, P. P. Jayaraman, R. Ranjan, "Cloud Resource Scheduling, Monitoring, and Configuration Management in the Software Defined Networking Era." TC-CPS Newsletter 3 (2017): 4-8.

Under Review

1. **Y. Li**, O. Almurshed, O. Rana, D. N. Jha, P. Patel, P. P. Jayaraman, and R. Ranjan, "A Fault-Tolerant Workflow Composition and Deployment Automation IoT Framework in a Multi-Cloud Edge Environment." IEEE Network, IEEE.

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Rajiv Ranjan, for his guidance and leadership throughout the years. I am incredibly grateful for his support, belief in me, and for the knowledge, wisdom and vision he has passed on to me. Rajiv has taught me to look at the wider implications of my research and has always had a positive outlook on what I have achieved. Furthermore, I wish to thank Prof. Paul Watson and Dr. Ellis Solaiman, who have been demonstrative and generous in their guidance and support in my PhD study. I am also very grateful to Dr. Prem Prakash Jayaraman (Swinburne University of Technology), Dr. Charith Perera (Cardiff University), Dr. Graham Morgan (Newcastle University) and Prof. Omer Rana (Cardiff University) for their guidance and valuable suggestions on my research.

I would like to thank my lab-mates and colleagues: Awatif Alqahtani, Ayman Noor, Bin Qian, Deepak Puthal, Devki Nandan Jha, Fawzy Habeeb, Jedsada Phengsuwan, Jie Su, Gagangeet Aujla, Khaled Alwasel, Nipun Balan, Umit Demirbaga, Zhenyu Wen; and my friends in Newcastle University: Bowen and his wife Min, Conghao, Jiajie, Yulong and his girlfriend Huan, Zongyin and his wife Shanshan for their kind help, support and company. I would like to thank my friends in China for their support and encouragement. I am grateful to many graduate students I met throughout the years in the School of Computing. I learnt something new from each one of them. I would like to thank Osama Almurshed from Cardiff University with his generous help on my research.

Lastly, I would like to thank my mother, Lazhi, my father, Jing, my mother-in-law, Lei, and all members of my family, who have been continuous with me throughout these years, especially my mother. Particularly, I would like to express my utmost thanks to my beloved wife, Die, for her continuous support, company, and patience on this long journey.

Contents

List of Figures

List of Tables

1

Introduction

Contents

Introduction

The Internet of Things (IoT) is the idea of things (devices) that are locatable, readable and recognisable and controllable through using the Internet [1–3]. It has become an important component of leading software development and an integral part of our daily life. In IoT, a number of things and resources (edge and cloud) combine to provide services that form the basis for many different types of applications that are commonly referred to as smart (e.g., smart healthcare, smart homes, smart buildings, smart manufacturing, smart agriculture, smart traffic). "Smart" is a way of categorising those applications where users can discover, query and employ different IoT entities on-demand in real-time without such entities requiring further human intervention in their development to achieve the desired outcomes. Manufacturers have developed numerous entities for use within IoT infrastructures. These entities are not only physical, such as sensors and actuators, but also virtual, like social media (e.g., Facebook, Twitter, MySpace). The heterogeneous large-scale data from such physical and virtual entities (e.g., [4], [5], [6], [7],) raises a challenge of unified resource configuration knowledge representation and high performance data processing [8]. Therefore, we need to incorporate the digital and physical worlds in IoT ecosystems. In order to allow this level of interoperability, it is important to define the services supplied by these physical and virtual entities in a unified way [9].

Flood-PREPARE [10] is a real-world IoT application for city rainfall analysis and flooding prediction. It allows real-time geographically precise warning for surface water flooding and continuous situational awareness and intelligence. To deploy such real-world IoT system, a range of resources from CCTV, sensors to Raspberry Pi, router and cloud virtual machines, GPU servers, etc. is required to coordinate extensive scale data for intellgent purposes. Due to the large data size, resource heterogeneity, model complexity and real-time processing requirements, it is a challenge to design and represent such IoT applications in an abstract way. More specifically, according to various domains, concepts and infrastructures of IoT application, it is hard to develop abstract methodology.

IoT applications can be modelled as a directed acyclic graph (DAG) with data trans-

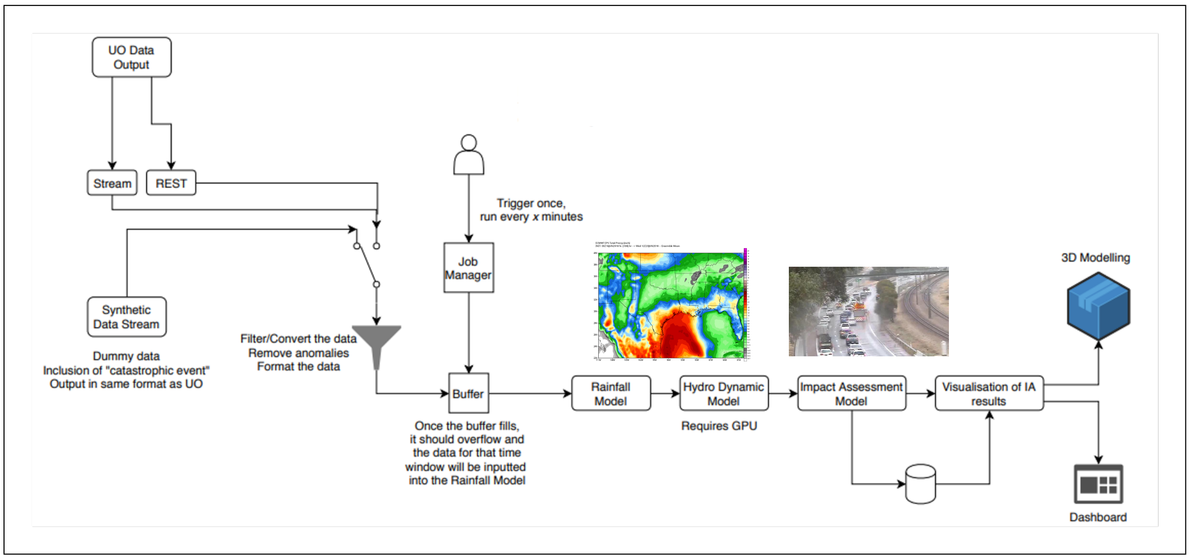

UO Data Output
Stream
REST
Synthetic Data Stream
Dummy data
Inclusion of "catastrophic event"
Output in same format as UO
Filter/Convert the data
Remove anomalies
Format the data
Trigger once, run every x minutes
Job Manager
Buffer
Once the buffer fills, it should overflow and the data for that time window will be inputted into the Rainfall Model
Rainfall Model
Hydro Dynamic Model
Requires GPU
Impact Assessment Model
Visualisation of IA results
3D Modelling
Dashboard

Figure 1.1: Flood-PREPARE system DAG

formation tasks as its nodes and data flow dependencies (or control flow dependencies) as its vertices. A simplified DAG of Flood-PREPARE is shown in Figure 1.1. To be most useful in a modelling sense, we need an IoT workflow pattern that can be general enough to define any IoT application. Representing a generic IoT workflow pattern is intricate because of the heterogeneity of IoT infrastructure (IoT device, edge device, and cloud), coupled with application dependency on infrastructure and variability of data. Identifying a set of reasonable IoT workflow activity patterns (provisioning an abstraction understandable to the engineer yet maintaining its usefulness for composition purposes), which covers IoT data transformation tasks and workflow activities, is the primary challenge in IoT workflow representation and composition. Addressing IoT application users' subjective and objective opinions with abstract understanding of IoT workflow instances remains a challenge for a more holistic approach.

Due to the rapid increase in IoT applications, the demand for fault-tolerance from infrastructure and software perspective has also expanded. In order to accomplish fault-tolerance for IoT application, such applications need to detect and recover emerging faults under the specific quality of service constraints, such as latency, etc. Therefore, developing a comprehensive fault-tolerant IoT system which offers self-detection and automatic recovery to increase application reliability and usability is also a challenge.

1.1 Research Questions

According to the research challenges discussed in the previous text in terms of IoT, Edge Datacenter (EDC) and Cloud Datacenter (CDC), it raises the question of how to deploy and configure IoT data processing activities across both edge and cloud datacenter. Thus, this thesis focus on answering the following research questions:

- *Q1: How to represent and discover IoT devices, EDC and CDC resources?*
- *Q2: How to map, execute and deploy data processing workflows across three layers based on demand requirements?*
- *Q3: How to increase workflow system reliability, availability, safety and maintainability in an IoT environment?*

To answer Q1, how to represent IoT devices, EDC and CDC resources need to be taken into account. In IoT applications, a large number of IoT devices need to be represented as virtual entities. It will benefit data analysis and operation afterwards. However, different embedded sensors and actuators are geographically distributed; also, one cloud resource provider cannot satisfy all the consumers' requirements. Therefore, proper devices from IoT environments and resources from edge and cloud need to be discovered and represented based on different demand and requirements.

Modelling IoT devices and EDC and CDC resource heterogeneity is another big challenge in representation. Heterogeneous configurations in IoT devices (e.g. location, temperature alarm, data transformation frequency of temperature sensors), edge datacenter (e.g. port configuration, IP address, certificate authority setting of a gateway), cloud datacenter (e.g. memory, IP address, storage of VM) are considered complex. Moreover, existing configuration management techniques are rarely transparent and adaptive to IoT applications, and the complexity of configuration management forces consumers to gain expertise in multiple configuration management knowledge domains.

For Q2, most IoT applications are modelled as data transformation workflows. Multiple heterogeneous data analysis computational and programming models that realise

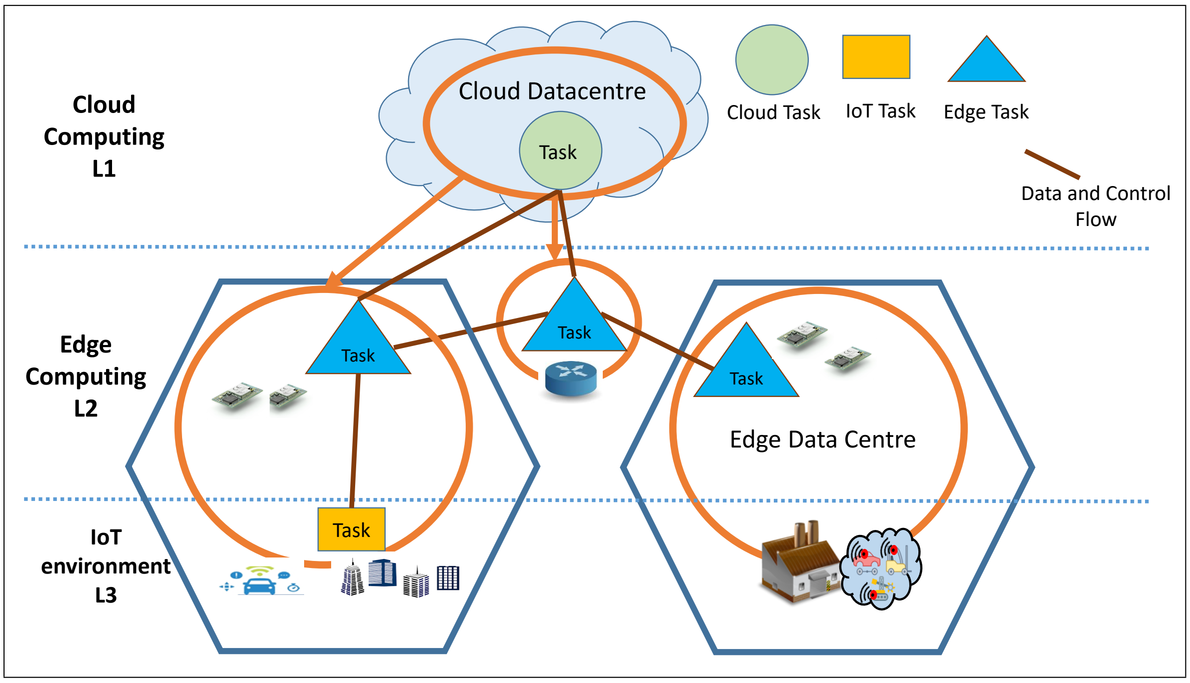


Figure 1.2: Data Transformation Tasks in Layered IoT Application

various data transformation tasks are shown in Figure 1.2. Data transformation procedures happen in the following three stages: between IoT devices layer and EDC layer; between IoT devices layer and CDC layer; between EDC layer and CDC layer. This new framework should enable an easy deployment interface where data transformation can be easily defined. Additionally, a composition technique which supports the composition of workflow crossing transformation in different layers needs to be considered.

The mapping problem is complicated by uncertainty in modelling the performance of workflow activities on both CDC and EDC resources. These resources have heterogeneous hardware and virtualisation configurations. Deploying and executing such workflow applications in different IoT environments is difficult as there is no unified method or programming tool to perform deployment in a variety of cloud providers and edge devices. Considering hardware and software heterogeneity, a more general and unified workflow deployment platform which can avoid the complexity of edge and cloud resources from customers needs to be designed and developed.

In Q3, the importance of IoT systems in terms of availability, safety, reliability and maintainability increase significantly due to the rapid development of IoT. Reliability is considered as a primary aim to implement for IoT applications concerned with

Quality of Service. Reliability is threatened by the occurrence of failures where an IoT system can hardly offer potential services. In an IoT environment, applications are excepted to continuously provide reliable services and features which put fault-tolerance mechanism as priority. IoT safety refers to the capability to secure or prevent the IoT system from causing an unacceptable risk of injury or physical damage. On the other hand, IoT safety can also be considered to secure the continuous IoT services providing from any unpredictable faults with both hardware and software. As a result, a framework which can efficiently execute a fault-tolerant IoT workflow application is required.

1.2 Research Contributions

In this thesis, I make principal contributions as following:

1. My first contribution is proposing a unified conceptual model which captures the resource configurations in IoT environments. I design a support recommender system (IoT-CANE) for the recommendation of resource configuration in IoT using SQL-based relational semantics and procedures. I also develop a service interface that converts simple context information captured from users to optimal IoT resource configurations to map users' requirements supported by an incremental knowledge acquisition based IoT resource configuration knowledge base. This main contribution is illustrated in Chapter 3.

2. My second contribution is proposing IoT workflow activity abstract patterns that can capture IoT user requirements. I also design and develop an IoT workflow composition system leveraging an analytic hierarchy process based multi-level composition algorithm. This contribution is presented in Chapter 4.

3. My third contribution is proposing a fault-tolerant IoT system to provide reliable workflow application execution, automation deployment and infrastructure recovery. I present a novel fault-tolerance IoT model and illustrate the details of a fault detection and infrastructure level recovery mechanism. A real-world IoT application, called Flood-PREPARED, is evaluated by the proposed fault-tolerant

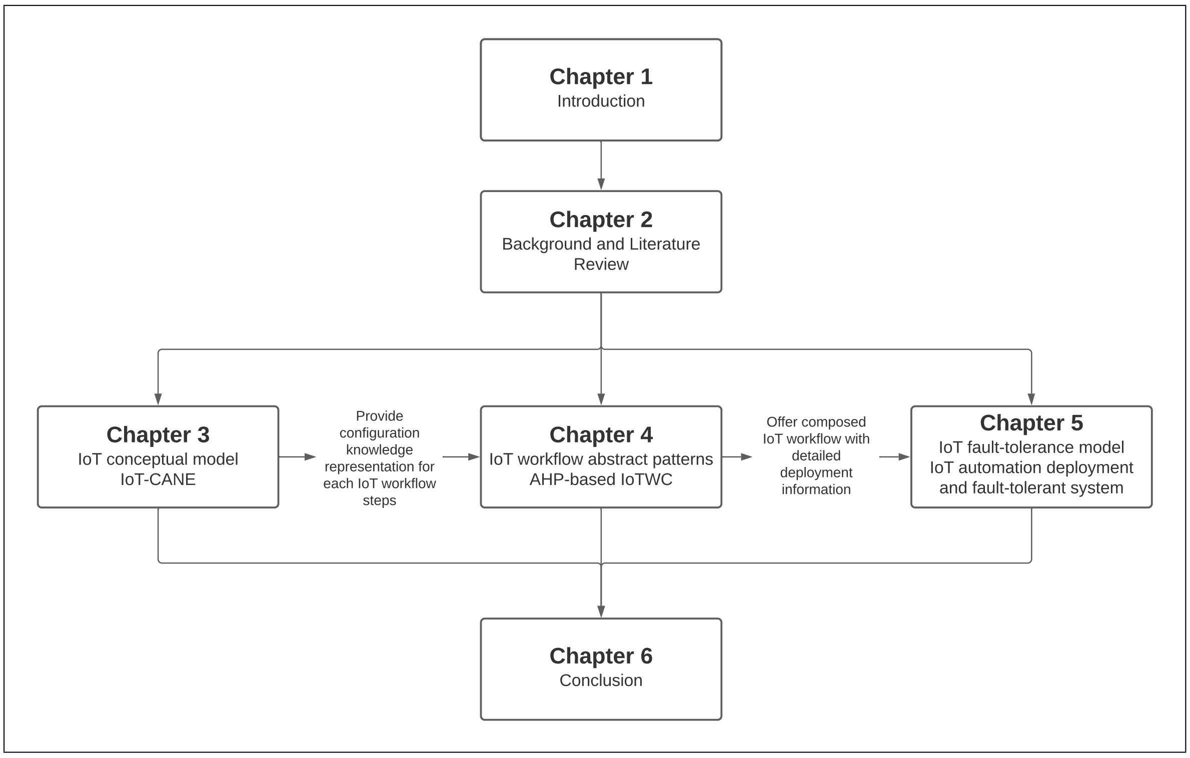

Chapter 1
Introduction
Chapter 2
Background and Literature Review
Chapter 3
IoT conceptual model
IoT-CANE
Provide configuration knowledge representation for each IoT workflow steps
Chapter 4
IoT workflow abstract patterns
AHP-based IoTWC
Offer composed IoT workflow with detailed deployment information
Chapter 5
IoT fault-tolerance model
IoT automation deployment and fault-tolerant system
Chapter 6
Conclusion

Figure 1.3: Chapter Structure

IoT system with results analysis. This contribution is discussed in Chapter 5.

1.3 Thesis Structure

This thesis is structured, as shown in Figure 1.3, and the details are as follows:

Chapter 1 describes the general background information and motivation behind the topic and illustrates the research questions and main contributions of this research.

Chapter 2 presents the detailed background material and state of the art regarding the overall topic in this thesis.

Chapter 3 presents IoT-CANE (Context Aware recommendatioN systEm) which embodies a unified conceptual model capturing configuration, constraint and infrastructure features of cloud/edge together with IoT devices. The performance and acceptance of the proposed system is investigated by a user case study. Based on the study results, a range of criteria were satisfied by the participants.

Chapter 4 proposes an IoT workflow composition system (IoTWC) to allow IoT users to pipeline their workflows with proposed IoT workflow activity abstract patterns. IoTWC leverages the analytic hierarchy process (AHP) to compose the multi-level IoT workflow that satisfies the requirements of any IoT application. The proposed IoTWC is evaluated on a user case study to evaluate the coverage of IoT workflow activity abstract patterns and a real-world scenario for smart buildings.

Chapter 5 proposes a fault-tolerant IoT workflow execution and automation deployment system. The proposed system enables a user to compose any IoT application with defined Quality of Service (QoS) parameters, then automatically set up the infrastructure according to the recommended configurations. Meanwhile, this system utilises fault detection and recovery mechanism to automatically detect and recover infrastructure level faults to increase application availability and reliability.

Chapter 6 summarises the conclusion of this thesis and describes future research directions in a related area.

2

Background and Literature Review

Contents

Summary

In this chapter, I present the background information regarding the overall topic. First, I describe the general Internet of Things (IoT) and related literature. Next, I present the configuration management and automation deployment in the IoT area. Last, I discuss the concept of workflow composition and orchestration. At the same time, academic research and industrial platforms in relevant topics are illustrated.

2.1 Internet of Things

With the diverse availability of computation and communication provided by cloud and edge systems, Internet of Things (IoT) generates a new way to visualise the interaction between physical and computation systems. In addition to computation and communication technology, IoT also depends on control systems, electronics and electrical engineering, chemical and biological advancements and other new design technologies to give a better interaction among these technologies. The rise of IoT technology revolutionises our way of living by influencing society in numerous ways, e.g. smart home, smart traffic, smart city, smart shopping, smart healthcare, smart agriculture, etc.

IoT is defined as an interdisciplinary approach that combines computation, communication, sensing and actuation of cyber systems with physical systems to perform time-constrained operations in an adaptive and predictive manner [11–13]. Here, feedback loops are associated with the physical system that helps embedded computers and networks to control and monitor physical processes. This helps in evolving the design technique based on the previous design model and feedback from the physical system, which results in the system being more reliable, robust and free from any previous error condition. IoT is an overarching concept having a number of branches that describe similar or related concepts. These include Cyber-Physical Systems (CPS) [14], Industry 4.0 [15], Machine-to-Machine [16], Smart City (Smart Anything) [17], etc. CPS is considered to be similar to IoT due to sharing similar architecture. However, the main focus of IoT is at the smart device level, whereas CPS emphasises the physical system.

There are three main components of CPS: cyber component, physical component and a

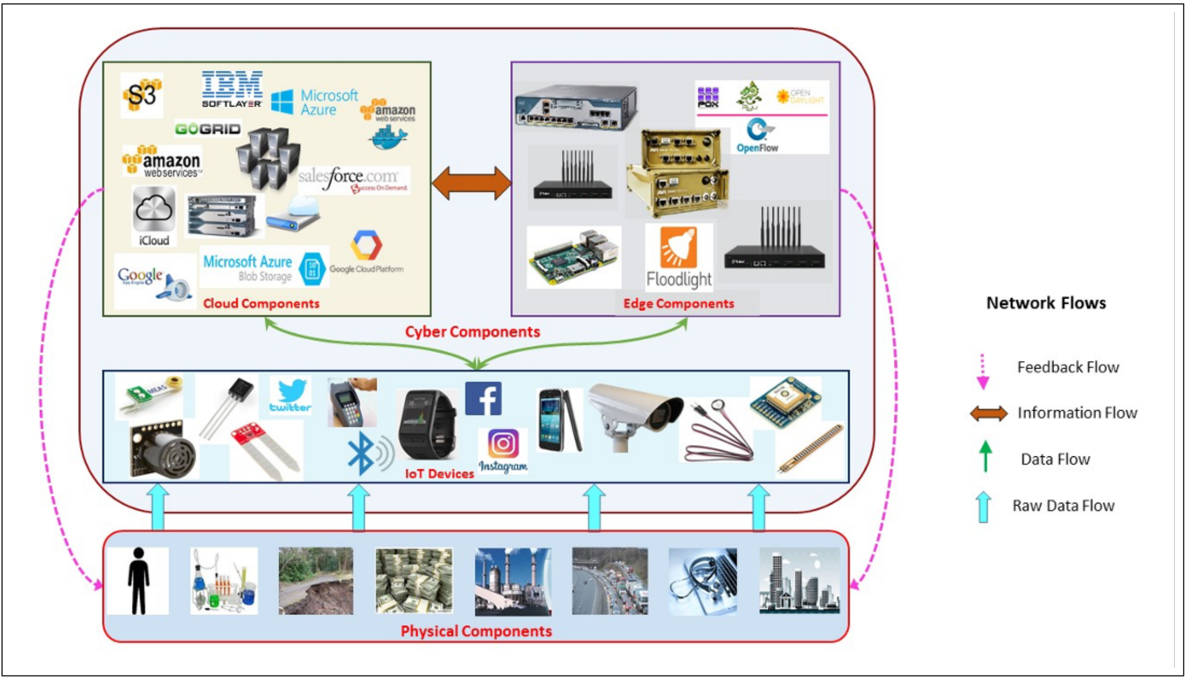


Figure 2.1: IoT/CPS System Components

network component. The cyber component consists of cloud, edge and IoT devices. IoT devices act as a bridge between the physical and cyber components. For implementing the desired solution, data is collected from diverse physical sources (e.g. environment, transport, communication, business transactions, healthcare system, education system, social media, etc.) by using smart IoT devices (e.g. sensors, cameras, log files, etc.) Increasing numbers of devices are continuously connecting to IoT/CPS systems to provide broader coverage of physical conditions. Gartner predicts that up to 100 billion devices will be connected to the IoT/CPS system by 2025 [18]. This data can be extracted, filtered and processed in many ways by IoT devices, edge and cloud datacenters.

2.1.1 IoT/CPS Architecture

IoT/CPS are mainly composed of three components, namely cyber component, physical component and a network component. Network components interlink the cyber and physical components for transfer of data and control. The interaction between all the components is shown in Figure 2.1.

Physical Component This component of IoT/CPS does not have any computation or communication capability. It includes chemical processes, mechanical machinery, biological processes and human aspects. Physical components create the data that mush be processed in real-time to operate and control various activities. These components generate data, which is highly concurrent and dynamic.

Cyber Component This component of IoT/CPS is responsible for the collection, processing, reporting and controlling of the physical components in IoT. It is very challenging to manage the dynamic and concurrent data produced by physical components. The cyber component consists of three sub-components: IoT devices, edge datacenter components and cloud datacenter components.

- *IoT devices*: These devices are highly integrated with the physical components to capture their activity. There are potentially millions of distributed IoT devices capturing raw physical data. Sensors (e.g. Temperature sensor, Humidity sensor, Motion sensor), actuators (e.g. Zigbee), mobile phones, cameras, etc. are the most common examples of IoT devices. Social media to capture humans' physical activity. A variety of devices can captures raw data in their native format. After capturing the data, IoT devices send it to the edge or cloud datacenter for further processing and storage. Some new IoT devices are battery-driven and can have limited capability.

- *Edge Datacenter (EDC)*: The EDC is the collection of smart devices capable of performing functions like storage or computation of diverse data at a smaller level. Smart gateways (e.g. Raspberry Pi, esp8266), Network Function Virtualisation Devices (e.g. OpenFlow, Middlebox) Software-Defined Networking, mobile phones, etc. are some common examples of EDC [19]. Each EDC device can perform specific functions, e.g. collecting data from various sensors, some pre-processing, short-term data storage and processing and finally routing the data to the cloud datacenter. The EDC also receives commands from the backend and routes it to the specific device. The EDC is also involved in ensuring secure communication by providing authentication and authorisation of IoT devices.

- *Cloud Datacenter (CDC)*: The CDC consists of all private, commercial and public cloud providers providing software, platform, storage, etc. in the form of services. These cloud providers are distributed in different geographical regions and can be accessed ubiquitously on demand. The CDC is considered to have unlimited storage and processing power. The physical datacenter resources are virtualised and provided to the user in a pay-per-use manner. Hypervisors (e.g. Xen, KVM, Virtual Box) and Containers (e.g. Docker, LXC) are two of the most common methods of virtualisation used in a cloud datacenter. The CDC creates multiple virtual machines (VMs or containers) over the physical machine that are isolated from each other and are allocated to different jobs [20, 21]. Many cloud service providers have a variety of virtual machines available with different configurations and costs. These machines can be used for massive data storage and processing activities. The virtual environment is selected on the basis of various QoS parameters, e.g. completion time, cost, availability, security, etc. [22].

Network Component This component of IoT is involved in all communication either between physical components and cyber components or among the cyber components. The raw data is captured from physical components by using IoT devices. The IoT devices then send the data to either the edge or the cloud or both. Information is transferred between the edge and cloud as required. Finally, cloud and edge devices send control and feedback to the physical devices. The factors affecting the network communication are network bandwidth, topology, contention, etc.

2.1.2 Data Analytics in IoT

The raw data captured from different physical devices is stored, analysed and processed in different ways to achieve the desired result. Data analytics refers to all the activities happening with the data after entering the CPS, including storage, processing, transfer, etc. [23]. These data analytic activities are performed at different layers (IoT devices or edge or cloud datacenter) depending on the requirements of the application (e.g. access methods, infrastructure support, hardware or software requirements). IoT devices are programmed to perform only specific functions, whereas edge and cloud datacenters

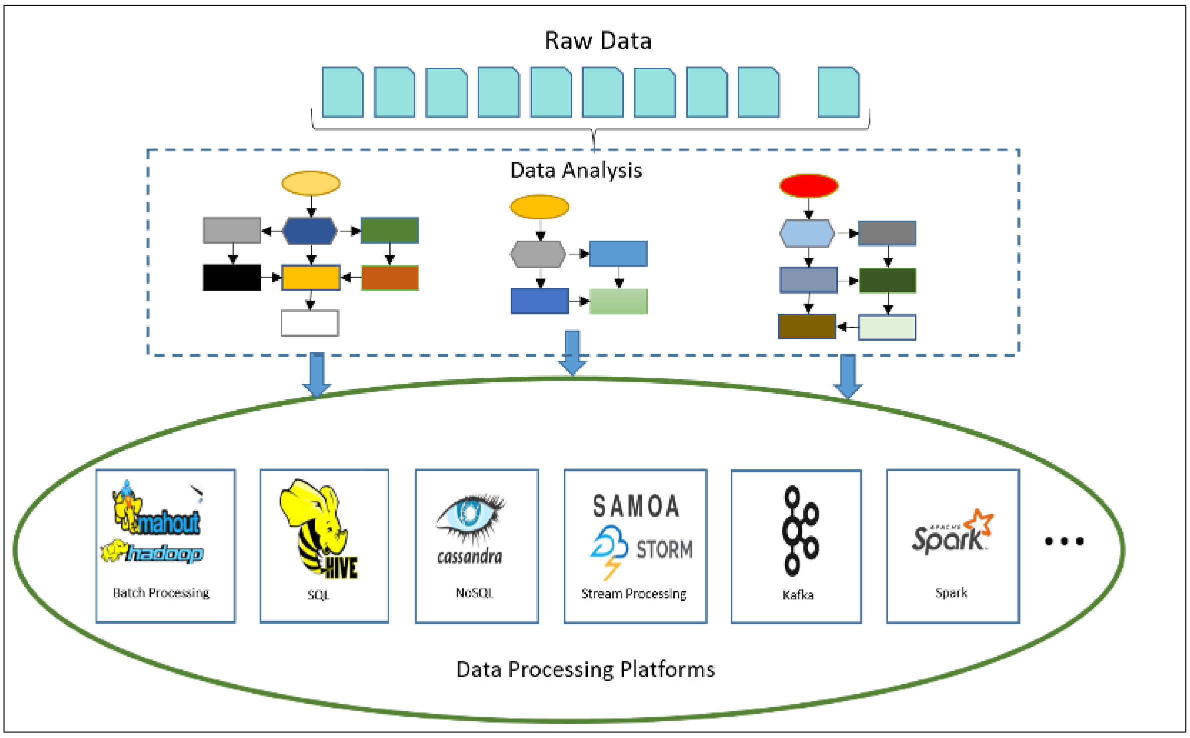


Figure 2.2: Data Analytic Operation

can be configured according to the application requirement [24]. The edge devices have small processing and storage capacity and normally use containers for creating virtual machines, whereas a cloud datacenter has enormous capacity and uses both VMs and containers for deployment.

The data captured from different sources is of large size, divergent type (e.g. text, image, audio, video) and is captured at variable speed. This data is either structured or unstructured. The complexity of the data makes it challenging to store and process it. Various existing programming models like batch processing, stream processing, SQL, NoSQL are involved in the storage and processing task for this data. Figure 2.2 shows a schematic diagram of data analytics where the raw data is converted to the workflow (representing the data and control dependency), which is then deployed on different data processing platforms including Hadoop, Hive, Storm, Kafka, Cassandra, etc. These data processing platforms are running on virtual machines/containers inside either edge or cloud datacenters.

2.2 Configuration Management and Deployment Automation

IoT solutions are typically application-specific and are deployed and configured on the basis of hardware heterogeneity (sensors, actuators, gateways, SDN controllers, datacenter, etc.), communication protocols (standard or specific, connectionless or connection-oriented, etc.), data processing models (batch processing, stream processing, etc.) and data storage models (SQL, NoSQL, etc.). The application environment differs from application to application, and there is no standard service management procedure available for all applications. Managing and handling such systems requires extensive knowledge of all the integrant technology, which is not possible for a user as they are considered to be either non-technical or with little technical knowledge. There is, therefore, a requirement for an autonomic system that performs all the configuration management and deployment automation procedures for a user. A user only needs to provide the requirement specification, and necessary constraints using the interface (e.g. command-line interface (CMI) or application program interface (API) or software-defined kit (SDK)) and the remaining processes are carried out automatically by the system.

Currently, numerous frameworks are available that can automate the deployment and configuration functionality of cloud or edge devices; however, these tools are not able to satisfy the complex dependency requirements of IoT. Due to the underlying heterogeneity of IoT infrastructure [25], these tools can only be used for automation of an individual layer of IoT application. The remaining service management tasks such as sensor and gateway configuration, different drives and package installation, various decision-making, etc., are handled manually in each particular case.

The motivation for this section comes from the challenges we have experienced while deploying smart solutions in different application domains. In this section, I discuss the problem of configuration management and deployment automation of IoT systems. The challenges of configuration management and deployment automation are discussed in terms of dimensions. I provide a brief overview of some existing commercial and open-sourced tools for configuration management and deployment automation and show that these tools are not completely suitable for IoT solutions.

The rest of this section is organised as follows. Subsection 2.2.1 discusses the problems related to the specification of configuration and deployment automation in IoT. I also present the dimensions of configuration management and deployment automation that affect this management process. In Subsection 2.2.2, I review some popular configuration and deployment frameworks used for cloud and edge environments. Subsection 2.2.3 evaluates the presented tools in terms of the dimensions identified in Subsection 2.2.1.

2.2.1 Concept and Dimensions of Configuration Management and Automation Deployment

Configuration management is a method that performs various system operations for a user, handling the entire system configuration and keeping track of files and packages [26]. However, the question is, why should we need configuration management? The answer is simple as the configuration management technique provides a simpler and faster application deployment with higher accuracy, flexibility and fault tolerance than manual methods. Manually configuring thousands of applications is very time consuming, with an increased chance of errors. If there is a software update, it is very tedious to update all the systems manually. There is a high chance of inconsistency if any system is not updated correctly. If there is some problem in the current version, rolling back to the previous version is more troublesome as no previous state information is stored.

All these problems can be overcome by using a configuration management system. By writing only one command, the application can be deployed or updated in as many systems as required, which makes deployment very simple, fast and flexible. As the configuration is performed automatically, there is a significantly reduced chance of error or inconsistency. The previous state information is stored by the configuration management system, which makes rolling back very easy in the case of some problem arising. Any default or error condition can be easily rolled back, making the system more fault-tolerant.

2.2.1.1 Configuration Management and Deployment Automation in IoT

Consider an example of a smart home in an IoT system. My phone can tell the heater or air conditioner to turn on and make the house warm/cool before I reach home, the virtual assistant (e.g. Gatebox, Amazon Alexa) could advise me to take an umbrella before I leave for the office and many other things. It looks very simple, but the processing and control of these operations are very complex. Sensors and actuators are embedded in the physical devices such as refrigerators, air conditioner, heating and lighting devices so that they can communicate with a central controller entity. The embedded sensors are continuously capturing the raw data, e.g. temperature of the room for the air conditioner or heater, weather information for updating about rain, etc. The decision about switching the heater or air-conditioner on is made by analysing the GPS data from the phone and the temperature sensor data from the house along with the stored data about the time taken to make the house warm/cool. The decision process is performed in either the edge or cloud datacenter as the IoT devices do not have that much capacity to store and process diverse data. Different IoT devices have different communication standards, which complicates communication with each other. Edge devices (e.g. Gateway) can eliminate this problem as they can easily be installed to communicate with different devices. These edge devices receive the data from different IoT devices and operate according to the demand of the application. For real-time constraint applications, the edge device performs some data analytic operations and notifies the IoT devices to perform accordingly while, for other applications, it extracts the data and sends it to the cloud for further storage and processing. The heavy data storage and processing are performed in the cloud, and the result is sent back to the IoT devices via edge devices. There may be intermediate information exchange between cloud and edge devices depending upon some factors like resource availability, time constraints, etc. The actuators embedded in the physical devices can react and respond according to the output of the edge or cloud. Feedback is also sent from the cloud or edge device that supervises the physical devices for self-configuration and self-adaptation.

To perform all these operations, a user would need to be expert in all the involved technology so that he/she can deploy and manage the complex requirements of IoT

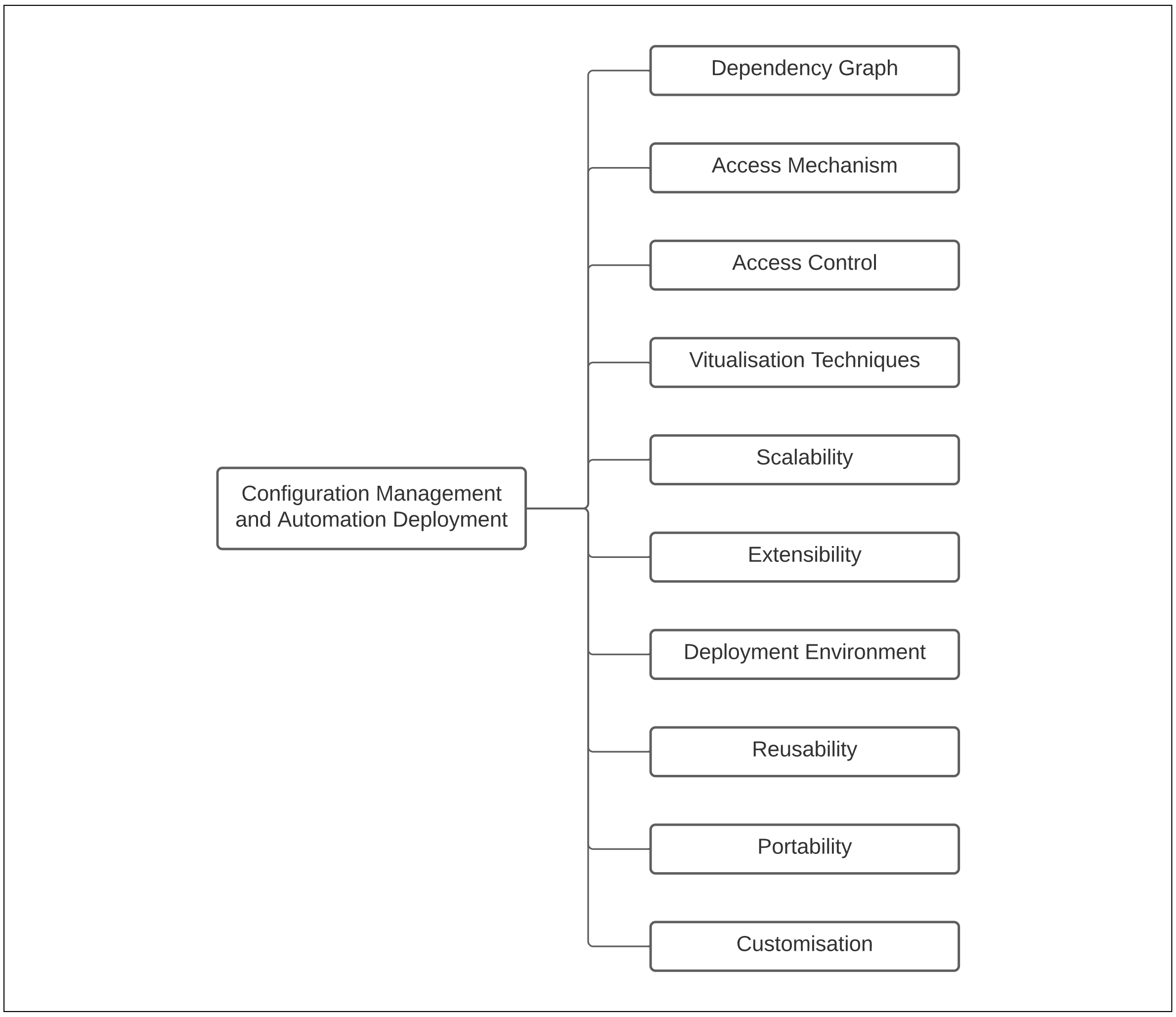


Figure 2.3: Dimensions of Configuration Management and Automation Deployment

applications. As the IoT application uses IoT devices, edge and cloud, a user must know how to configure and manage data in all these environments. It is not enough to select an optimal cloud resource or edge resource when providing the best services. The cloud, edge and IoT devices must be synchronised together to provide better service with maximum resource utilisation. Given the variety of devices at each level, as shown in Figure 2.1, it is not possible to manually configure the whole system by writing one script that manages everything. In addition to this, the continuous update and upgrade of the execution environment make the management process more complicated. A solution is required that performs all these operations automatically. In an automated environment, the user only needs to state the requirements and constraints, and the remaining processes are performed automatically.

2.2.1.2 Dimensions of Configuration Management and Deployment Automation

The complexity of data analytic activities in the cloud, edge and IoT environment makes the configuration management and deployment automation process very challenging. To facilitate this process, we present the technical dimensions that provide an intuitive view of the factors affecting the configuration management and deployment automation in IoT applications. The dimensions are depicted in Figure 2.3 and are explained in this section.

Dependency Graph A dependency graph is a directed acyclic graph that represents the dependencies of several nodes (objects/resources) on each other. The ordering relation of each node can easily be extracted from the dependency graph. One can easily and explicitly represent the data and control flow among the nodes by using a program dependency graph (PDG). The dependency between constituent entities/nodes can be easily analysed by using a PDG, which can be used to support user interaction, parameterisation of models, optimisation decisions, consistency checks, easy debugging and other operations. Dependencies are broadly classified into two categories: data dependency and control dependency. Data dependency shows the flow of data being computed by one node that is used by other nodes, whereas control dependency represents the ordered flow of control in the program. Any change in the dependency can easily be represented by adding, removing or reconfiguring them on the PDG.

It is straightforward to represent the complex dependency and requirement specification of IoT by using the dependency graph. Here, the node represents an application/service, whereas the edges represent the dependency, i.e. how the nodes are dependent on each other. This is also helpful for creating a similar application as we can reuse the same dependency graph with some modification.

Access Mechanism Access mechanism signifies the methods of interaction with the services provided by the IoT system. This is very important as it provides an abstraction for different types of users, e.g. application developers, technical experts or DevOps managers accessing from different types of device, e.g. personal computers, mobile phones, tablets and facilitates easy interaction with the IoT. There are numerous access mechanisms available, e.g. command-line interface (CLI), application program interface (API), software development kit (SDK), graphical user interface (GUI). These mechanisms have different properties; for example, GUI is easier to use but has significant delay as compared to a command-line interface. The choice of access mechanism depends on various factors such as application support, device support, user technical knowledge, etc.

Access Control In information technology, access control is a process by which users are granted access and certain privileges to systems, resources or information in a computing environment. However, applications in cyber-physical systems, unlike traditional applications, usually do not have well-defined security perimeters and are dynamic in nature. Therefore, traditional access control policies and mechanisms rarely address these issues and are thus inadequate for IoT. Access control for IoT depends on the following factors: trustworthiness of entities; environmental context, and application context. In terms of trustworthiness of entities, this is important in IoT because IoT has no well-defined security perimeters (interactions between entities may be unknown in advance). The overarching theme between two types of access control (physical and cyber) is a notion of trust. Environmental context, such as location and time, is also a crucial consideration in access decisions of IoT. Access control models must take into account environmental factors before making access decisions. For the application context, it depends on the data obtained from sensors and other devices.

Access control systems perform authorisation identification, authentication, access approval, and accountability of entities through login credentials, including passwords, personal identification numbers (PINs), biometric scans, and physical or electronic keys.

Extensibility Extensibility is the ability of a system to be easily extended or expanded from its initial state. It is an essential characteristic of any software, application or programming language, which makes it adaptable for execution in a frequently changing environment. Extensibility can be supported by add-ons, plug-ins, packages or hooks that add some additional functionality or by explicitly adding macros or functionality directly to the applications. Ruby, Lua, etc. are typical examples of extensible programming language. At the same time, Eclipse is a typical example of an extensible application which provides a variety of add-ons (available offline or through a market place) that can easily be integrated with the existing application. The technology is changing at a fast rate, so an application must be extensible so it can be adapted for any new environment.

Customisation Customisation or personalisation is the characteristic that allows a user to customise an application based on their specific requirements. The requirements of each user are unique, and so do not necessarily fit with the other user applications or a default application. If the system allows a user to customise their application, then it is better from both user and system perspective as all the requirements of users are satisfied. Targeted systems resources are used for reducing resource wastage.

Reusability Reusability refers to the reuse of existing software artefacts. There are some modules which are common in multiple applications. One way to achieve this is to define the same modules separately for each application, while the alternative is to define the module only once and reuse the same module multiple times. Many existing modules can either be used directly or be used after little modification. There are some essential requirements for software/products to be reused, including consistency, adaptability, stability, flexibility and complexity.

Tools like Docker have a storage repository (Docker Hub) that stores numerous existing containers. A user can easily pull the image from this repository. These images can either be used directly or easily updated to satisfy some specific requirements.

Deployment Environment The deployment environment is the system(s) responsible for the proper execution of an application. The environment provides all the necessary resources to start, execute and stop the application. Based on the resource type and configuration, the deployment environment can be categorised as an on-premise system, edge datacenter or cloud datacenter. Cloud datacenter is again divided into public, private or hybrid cloud. The application can be deployed based on different qualitative and quantitative QoS parameters, e.g. resource requirements, cost, deadline, security, etc.

Virtualisation Technique Virtualisation is the crucial concept of cloud computing that partitions the physical resources (e.g. compute, storage, network) into multiple virtual resources [20]. It allows multiple users to access the services provided by the cloud in an isolated manner. Two types of virtualisation are expected in a cloud perspective: hypervisor-based virtualisation (e.g. Xen, KVM, VMware) and container-based virtualisation (e.g. Docker, LXC). In hypervisor-based virtualisation, a hypervisor layer is either added directly on top of the hardware (Type 1 hypervisor) or on top of the host operating system (Type 2 hypervisor). However, each virtual machine must have a separate operating system. In container-based virtualisation, a container engine is added on top of the host operating system that is shared by all the containers using Linux features namespace and c-groups. Container-based virtualisation is considered to be lightweight due to the lack of a separate operating system for each container.

Scalability Scalability is the ability to accommodate an increasing workload by increasing the capacity of the system. It is an essential characteristic of an application, which determines whether the application can perform well with higher workloads. Scalability can be performed either vertically (scale-up) or horizontally (scale-out). The difference between these two approaches is the way they add additional resources with an increasing workload. In vertical scalability, the capacity of the node is increased by adding more resources (e.g. CPU, memory, disk). In contrast, in horizontal scalability, we can add many independent systems (equivalent to the existing system) parallel to the existing systems to satisfy the increased workload.

Scalability can be achieved through a software framework such as dynamically loaded plug-ins, top-of-the-line design interfaces with abstract interfaces, useful call-back function constructs, a very logical and plausible code structure, etc.

Portability Portability is the characteristic of an application that allows it to execute successfully in a variety of environments. Portability is a capability attribute of a software product, and its behaviour is manifested to a degree, and the degree of performance is closely related to the environment. The basic idea of portability is to provide an abstraction between system and the application, which is possible only if the application is loosely coupled with the system architecture.

Portability is critical as it can improve the software life cycle by allowing it to run on different domains. It also reduces the burden of redefining the same application for different environments, which increases the Reusability of the application.

2.2.2 Configuration Management and Deployment Automation Tools: State of the Art

The research community has developed several frameworks to automate the deployment and configuration management in the cloud and edge environments. Some of the frameworks can also have support for IoT applications. Users only need to specify the requirements and these frameworks perform the remaining operations automatically. Some popular frameworks and their functionalities are presented below.

Chef Chef [1] is a powerful tool for automating the infrastructure. It converts the infrastructure in the form of code called Recipe for automating the setup, configuration, deployment and management. It can be used for different types of application varying from web application to batch and stream processing.

There are three main elements of Chef: a server, a few workstations and many nodes. Any machine (physical or cloud) managed by Chef is called a node. Each node is installed with a Chef client that automates all the management operations on that node. Each client is registered with the Chef node before the configuration management. Chef uses a client-server architecture where each node runs a Chef client while the server is available to all the clients. The Chef server is a centralised entity that contains all the Cookbooks (a Cookbook is a basic unit of configuration management that defines a scenario with everything necessary to handle that scenario.), policies and

[1]https://www.chef.io

other management information. The client retrieves all the stored information on the server and pulls the relevant configuration data to automate the management of the node. The client uploads the runtime data with the respective scenario to the server so that some other node can use the information in the future. The workstation acts as a communication bridge between a client installed on a node and the Chef server. It runs a Chef development kit (ChefDK) that facilitates client and server interaction and also helps users to author, test and maintain Cookbooks on the workstation, which are then uploaded to the server. The programming language Ruby is used to author the Cookbooks.

Puppet Puppet [2] is an open-source configuration management tool that allows users to define the system resources and the state of a system (i.e. desired state) and performs all the operations to achieve that state from the current state. The system resources are described by the user using an easy-to-read declarative language or Ruby domain-specific language. Puppet works either by using a client-server architecture or as a stand-alone application.

In a client-server architecture, a server (also known as the master) is installed on one or more systems, and a client (also known as an agent) is installed on all the nodes to be configured. The agent running on a node communicates with the server and conveys all the desired information from the server. The agent first sends the node information to the server; the server uses that information to decide what configuration should be applied to that node. The master then sends the desired information to the agent, and the agent implements all the changes accordingly to reach the desired state. The Puppet agent can run either periodically to check the configuration of the node or can be configured manually to set the specific configuration.

The declarative nature of Puppet permits uses of the same resource declarations multiple times without any alteration to the result. Maintaining a centralised codebase is more manageable than distributed code, while also increasing productivity.

[2]https://puppet.com

Ansible Ansible [3] is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration and many other IT needs. It uses no agents and no additional custom security infrastructure, so it is easy to deploy – and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allows the user to describe automation jobs in a way that approaches plain English.

Ansible works by connecting to nodes and pushing out small programs, called "Ansible Modules" to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default) and removes them when finished.

Docker Docker [4] is an open-source platform that automates the deployment and management of applications in containers. It is the most popular Linux container management tool that allows multiple applications to run independently on a single machine (physical or virtual). It abstracts and automates the operating system level virtualisation on any machine.

Docker wraps the application with all its dependencies into a container so that it can efficiently be executed on any system (on-premise, bare metal, public or private cloud). Docker uses layered file system images along with the other Linux kernel features like namespace and cgroups for management of the containers. Other container management tools do not support the layered file system feature. This feature is handled by an advanced multi-layered unification file system (aufs) which provides a union mount for the layered file system. This feature allows Docker to create any number of containers from a base image, which are simply copies of the base image. Docker can easily be integrated with different infrastructure management tools such as Chef, Puppet, Kubernetes, etc.

Multiple Docker nodes can quickly be scheduled and managed by Docker swarm, which manages the whole cluster as a single virtual system. Docker swarm uses the standard Docker API to interact with the management tool (e.g. Docker machine) to assign

[3]https://www.ansible.com
[4]https://www.docker.com

containers to the physical nodes in an optimised way.

Kubernetes (K8s) Kubernetes [5] is an open-source platform used to automate the operations (e.g. deployment, scaling, management) of containerised applications across a group of hosts. Kubernetes API objects are used to describe the desired state of the cluster (e.g. information about the application, type of workloads, container images, network and disk resource requirements). The desired state information is set either by using Kubernetes CMI kubectl or by accessing the API directly. Kubernetes control plane checks the current state and automatically performs a series of tasks to match it with the desired state. Kubernetes control plane splits the cluster into two types of nodes, namely master node and non-master node. The master node is installed on one node, and it controls the whole system by running three processes: kube-apiserver, kube-controller-manager and kube-scheduler, while non-master nodes or Minions are installed on the other nodes, running two processes: kubelet (communicates with the master node) and kube-proxy (network proxy reflecting network services on each node). For the sake of high availability and fault tolerance, the number of master nodes is more than one, i.e. one is the primary master node, and the others are supporting master nodes (copies of the primary master node).

The Kubernetes object represents the abstract view of the state of the cluster. The necessary objects of Kubernetes are Pod, Service, Volume and Namespace. Two fields govern the configuration of an object's state: the object spec and the status, which describes the desired state and actual state respectively. The Pod is the smallest and simplest unit of the Kubernetes object model that represents a running process. It contains one or more co-located containers that share common resources (storage, memory and network). Based on the requirement specification of the user, Kubernetes creates and controls multiple pods.

Juju Juju [6] is an open-source universal modelling tool for deployment, configuration, management, scaling and maintenance of applications on physical servers and cloud environments. Juju provides a framework that allows users to define their requirements

[5]https://kubernetes.io
[6]https://juju.is

abstractly. Juju works a layer above the usual configuration management tools like Puppet, Chef, etc. and uses the services provided by these tools. It mainly focuses on the service delivered by the application rather than on the environment or platform on which it is running.

The whole mechanism of Juju is based on “charms”. Charms contain all the information regarding the deployment of an application. Charms can be written in any programming language or scripting system. Users can either create their charms, use any existing charms or update existing charms by merely adding some features according to their requirements. All the charms and their relationships are contained in a bundle that provides the full working deployment in one collection. A bundle file is easy to share among different users working in different environments.

Amazon CloudFormation Amazon Web Services Cloud Formation [7] is a free service that provides Amazon Web Service (AWS) customers with the tools they need to create and manage the infrastructure required to run on AWS. CloudFormation has two parts: templates and stacks. A template is a JavaScript Object Notation (JSON) text file. The file, which is declarative and not scripted, defines what AWS resources or non-AWS resources are required to run the application. For example, the template may declare that the application requires an Amazon Elastic Compute Cloud (EC2) instance and an Identity and Access Management (IAM) policy. When the template is submitted to the service, CloudFormation creates the necessary resources in the customer’s account and builds a running instance of the template, putting dependencies and data flows in the correct order automatically. The running instance is called a stack. Customers can make changes to the stack after it has been deployed by using CloudFormation tools and an editing process that is similar to version control. When a stack is deleted, all related resources are deleted automatically as well.

Terraform Terraform [8] is a tool for building, changing and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions. Configuration files describe the components

[7]https://aws.amazon.com/cloudformation/
[8]https://www.terraform.io

needed to run a single application or an entire datacenter. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform can determine what changed and create incremental execution plans. The infrastructures that Terraform can manage include low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.

Some of the key features of Terraform are:

- Infrastructure is described using a high-level configuration syntax. Additionally, infrastructure can be shared and re-used.
- Terraform has a planning step where it generates an execution plan. The execution plan shows what Terraform will do when it is applied. This avoids any surprises when Terraform manipulates infrastructure.
- Terraform builds a graph of all resources and parallelises the creation and modification of any non-dependent resources. Because of this, Terraform builds infrastructure as efficiently as possible, and operators get insight into dependencies in their infrastructure.
- Complex changesets can be applied to infrastructure with minimal human interaction. With the previously mentioned execution plan and resource graph, users know what Terraform will change and in what order, avoiding many possible human errors.

Cloudify Cloudify [9] is an open-source framework that allows you to deploy, manage and scale your applications on the cloud, and Cloudify aims to make this as easy as possible. The idea is simple: users model their application using a blueprint, which describes the tiers that make up their application, along with how to install, start and monitor each of these tiers. The blueprint is a collection of text configuration files that contain all of the above. Each Cloudify deployment has one or more managers, which

[9]https://cloudify.co

are used to deploy new applications (using the above blueprints), and continuously monitor, scale and heal existing applications.

Cloudify looks at the configuration from an application perspective – i.e. given a description of an application stack with all its tiers, their dependencies, and the details for each tier, it will take all the steps required to realise that application stack. This includes provisioning infrastructure resources on the cloud (compute, storage and network), assigning the right roles to each provisioned VM, configuring this CM (which is typically done by CM tools), injecting the right pieces of information to each tier, starting them up in the right order, and then continuously monitoring the instances of each tier, healing it on failure and scaling that tier when needed. Cloudify can integrate with these CM tools as needed for configuring individual VMs, and in fact, this is recommended as best practice.

TOSCA TOSCA [27, 28] is a new OASIS standard that specifies the meta-model for explaining the topology structure and management of cloud application. The structure is defined by a topology template (consisting of a node template and a relationship template) represented as a directed graph. Node type is defined separately to support reusability and is referenced by the node template whenever required. There are two methods for implementing TOSCA: the declarative method and the imperative method. In the declarative method, the user declares the requirements, and the framework automatically does everything, while in the imperative method, the management process is specified explicitly using Plans. Plans are the process models that define the deployment and management process and are represented using complex workflows. Open TOSCA is the runtime instantiation of TOSCA. TOSCA also provides a limited run time ecosystem for an IoT environment.

Calvin Calvin is an application environment that lets things talk to things [29]. It includes both a development framework for application developers and a runtime environment for handling the running application. The Calvin Platform is an attempt at a solution allowing developers to develop applications using clearly separated, well-defined functional units (actors) and per-deployment requirements. The platform then autonomously manages the application by placing the actors on different nodes (devices, network nodes, cloud, etc.) in order to meet the requirements, and later migrates them if changes in circumstances should so require. The platform enables decoupling application development and deployment from hardware investments by providing an abstraction layer for applications and establishing a standard interface to similar functionality, built up in an agile manner. In [30], the authors divided an application's life cycle into four separate, well-defined phases: Describe, Connect, Deploy and Manage. Then Calvin uses the actor model to provide abstractions of device and service functionality. With actors describing the processing blocks, they concisely express the data flow graph using CalvinScript. A Calvin runtime presents an abstraction of the platform it runs on as a collection of the capabilities this platform offers. One of the friendly properties of automated deployment based on capabilities and requirements is that many aspects of managing running applications are already in place and active.

2.2.3 Evaluation of Configuration Management and Deployment Automation Tools

This section evaluates the different tools discussed in Subsection 2.2.2 in terms of configuration management and deployment automation dimensions. The summary of evaluation is presented in Table 2.1 and Table 2.2 and is described below.

- Most of the tools have a dependency graph that helps to represent the resource definition and relationship except Ansible that does not have dependency graph support. Some of the tools (e.g. Puppet) use the dependency graph only for internal resource management. The dependency graph in Cloudify is represented using a blueprint that describes the logical representation of the application. TOSCA is the only tool that represents the topology specification along with the resource representation.

- In terms of extensibility, most of the tools are extensible, but how this is achieved is different. Some of the tools can be extended by the addition of plugins (e.g. Juju, Cloudify) while some use an extensible library (e.g. Ansible).

Table 2.1: Evaluation of Configuration Management tool (Part I)

Tools	Dependency graph	Access mechanism	Access control	Extensibility	Customisation
Chef	Yes	CDK (chef development kit), GUI (graphical user interface)	Mutual authentication, SSL	Yes	Yes
Puppet	Yes	RPC (remote procedure call), API (application program interface), GUI	Mutual authentication, digital signature, SSL	Yes (using faces API)	Yes
Ansible	No	CLI (command line interface), web user interface	SSH	Yes (extensive library)	Yes
Docker	Yes	CLI, Docker engine API	Linux namespace and cgroups, TLS (transport layer security) using plugins	Yes (via plugins)	Yes
K8s	Yes	CLI, API, REST based request	Client certificate, Password, plain/bootstrap, tokens	Yes (modular, pluggable, hookable, composable	Yes
Juju	Yes	CLI, GUI	Mutual authentication, SSH	Yes (using plugins)	Yes
Cloud Formation	Yes	CLI, management console, API	AWS IAM (identity access management)	Yes	Yes
Terraform	Yes	CLI, API	End-to-end authorisation	Yes	Yes (via plugins)
Cloudify	Yes	CLI, web user interface	Mutual authentication. SSL	Yes (via diamond plugins)	Yes
TOSCA	Yes(with topology information)	GUI	No specific client authentication method	Yes	Yes
Calvin	-	CLI, API	Mutual authentication	Yes	Yes

Table 2.2: Evaluation of Configuration Management tool (Part II)

Tools	Reusability	Deployment environment	Virtualisation technique	Scalability	Portability
Chef	Yes (using cookbooks)	On-premise, cloud, edge	Hypervisor, container	Yes	Yes
Puppet	Yes	On-premise, cloud, edge	Hypervisor, container	Yes	Yes
Ansible	Yes	On-premise, cloud	Hypervisor, container	Yes	Yes
Docker	Yes (using Docker hub)	On-premise, cloud, edge	Container	Yes (horizontally scalable using parallel and sequential mode), no autoscaling	Yes
K8s	Yes	On-premise, cloud, edge	Container	Auto scaling	Yes
Juju	Yes (using charms or bundles) (puppet, chef, Docker, etc. codes can also be included in the charms)	On-premise, cloud, edge	Hypervisor, container	Yes	Yes
Cloud Formation	Yes	Public cloud (AWS)	Hypervisor, container	Auto scaling	Compatible with all the services provided by AWS
Terraform	Yes	On-premise, cloud, edge	Hypervisor, container	Yes	Yes
Cloudify	Yes	On-premise, cloud, edge (with vCloud plugin)	Hypervisor, container	Yes	Yes
TOSCA	Yes	On-premise, cloud	Hypervisor, container	Yes	Yes (using TOSCA-based DSL)
Calvin	Yes (using Calvin actors)	Cloud, edge, IoT devices (mainly for IoT application)	Hypervisor, container	Yes	Yes

- Most of the tools support user customisation, but the way they allow customisation is very different from tool to tool.
- Except for TOSCA that does not have a specific access control mechanism, all other tools have their specific mechanism. Some tools (e.g. Chef, Juju, Puppet) use mutual authentication while others (e.g. Cloud Formation) use a cloud-specific identity access mechanism (IAM).
- Since most of the DevOps systems are based on UNIX or Linux system, most of the tools support command-line interface (CMI). Some of the tools (e.g. Puppet, Docker, Calvin) also support an application program interface (API) to manage applications across cloud, edge and IoT. A few of the tools (e.g. Chef, Juju, TOSCA) also support a graphical user interface (GUI) for easy accessibility.
- Most of the available tools (e.g. Ansible, Cloud Formation) are used for on-premise or cloud datacenter, but only a few of them can be used for edge devices (e.g. chef, Puppet, Docker). Among these tools, only a few of them (e.g. Juju) are used for infrastructure deployment. Calvin is the tool specially designed for IoT application deployment while recent advancements in TOSCA also support IoT. The remaining tools do not contain support for CPS applications.
- Docker and Kubernetes are the tools that support only container based virtualisation; the remaining tools support either hypervisor only or both hypervisor and container. The virtualisation technique is not relevant for tools like Chef, Puppet, Ansible, etc., as they are only responsible for configuration management of resources rather than virtualising them. Docker and Kubernetes have recently started to focus on infrastructure management.
- Reusability is an important concept, which is provided by almost all tools, using different methods. Chef uses Cookbook while Docker uses Docker Hub. Similarly, other tools use some centralised or distributed storage to reuse the artefacts.
- Most of the tools provide scalability, but only some of them (e.g. Kubernetes, Cloud Formation) provide automatic scaling. Similarly, portability is an essential requirement for a configuration that is provided by almost all the tools.

From the evaluation, it is clear that the presented tools automate the deployment and configuration management task in their specific ways. Frameworks such as Chef, Puppet, Ansible, etc. configure and deploy the infrastructure but do not virtualises the infrastructure. Tools like Docker and Kubernetes virtualise only in the form of containers, whereas Juju and Cloud Formation support both container and hypervisor-based virtualisation. Tools like TOSCA and Calvin have support for IoT devices along with cloud and edge environment, but they need further development. To provide a sophisticated solution for deployment and configuration management, we need a new tool that can satisfy all the dimensions holistically.

2.3 Workflow Composition and Orchestration

A workflow usually represents sequenced activity patterns that can be composed and orchestrated based on a business target. With the development of business process modelling and business process coordination, workflow technology becomes essential to deal with complex component frameworks and business interaction.

Workflow can be understood through several different perspectives [31], such as data-flow, control-flow, resource, etc. In the control perspective, a workflow illustrates activities and the execution order to enable workflow execution, such as sequence, parallelism, choice and join synchronisation. From the data perspective, the workflow can be divided into workflow data patterns, including data visibility, data interaction, data transfer and data-based routing [32]. When it comes to the resource perspective, it provides a low-level structure to the workflow in the format of human and resource roles.

The control flow provides an initial vision of a workflow specification, including the essential connections in business processes. Moreover, data flow offers a unified way to represent and utilise workflows with designed workflow data patterns.

In the rest of this subsection, I discuss the scientific workflow system management, composition, execution and provenance first. Then I move to IoT workflow composition and orchestration. Finally, I list related IoT workflow tools.

2.3.1 Scientific Workflow System

With increasing application complexity and distributed resources, a scientific workflow system has become a necessary tool to overcome such challenges [33]. The purpose of scientific workflow systems is to simplify and automate the data movement between different locations and activities in order to reduce the complexity of computational management for scientists.

A workflow in such systems represents an orchestration template consisting of specific scientific workflow activity patterns and dependencies between them. To perform a workflow lifecycle, the following five steps are considered: specification, composition, scheduling, execution and provenance [34]. First, a data model is specified and designed based on a particular goal to represent each component in a workflow application. Next, according to this data model, a comprehensive workflow can be composed to achieve the final goal with desired workflow activities. After the workflow is defined, it can be passed to schedule and execute on natural resources. Once execution finished, various registries can save and record all relevant provenance information to provide insights for future workflow design. I explain these five workflow lifecycle steps in detail in the following subsections.

2.3.1.1 Scientific Workflow Specification

Workflow specification aims to offer a unified and textual way of allowing workflow owners who can specify requirements to control and adjust the workflow step selection and composition. Workflow specification can also be understood as a data model design for a particular scientific application. As such, the workflow specification process is user-specific requirements driven. To specify user requirements, a unified workflow specification language is necessary in the workflow design stage to represent requirements that can be reasoned, validating properties of services and verifying services to be part of a workflow.

Holonic Multi-agent Systems (HMS) [35] is designed to address the dynamic of workflow services. This system is adopted for dynamic real-time re-configuration for service collaboration. A workflow adaptation problem (WAP) is analysed, and a solution based on contract net protocol (CNP) is proposed to solve WAP in the context of service composition based on HMS. A well-established workflow language, Petri Net [36] is employed in this work to represent the workflow.

Business Application Modeler (BAM) [37] is a tool-based approach for requirements verification and validation for business process models. The core component in this model is the formal specification which can be adopted for automatic verification and validation of process models. In this work, graph-based rules in Computation Tree Logic (CTL), Graphical Computational Tree Logic G-CTL [38] is used to specify the workflow requirements.

A concurrent Transaction Logic (CTR) [39] based formal framework is proposed for interaction modelling in virtual enterprises. Workflows are graphically presented as a Direct Acyclic Graph (DAG), representing activity coordination. In order to enforce constraints on a workflow or specific control flow, a set of CTR connectives are created.

SpecificatiOn Language fOr servIce compoSitions inTeractions (SOLOIST) [40] is a workflow specification language to formalise the interactions of services composition. It follows the Service Oriented Architecture (SOA) principles to enrich service-based application development by orchestrating third-party services to provide added-value applications. This service-based application has been widely used in enterprises, to develop their own SOA information systems.

Service workflow specification (SWSpec) [41] language is devised to specify workflow requirements mathematically relevant to service workflows. It provides a unified and formal method to allow requirements to be specified independently. The compliance checking is automatic due to the formal SWSpec. SWSpec captures the structural features of service workflows, such as tasks and the tasks execution locations, to represent the coordination of these tasks.

2.3.1.2 Scientific Workflow Composition

Workflow composition acts a crucial role in scientific workflow lifecycle. It allows users to define the workflow activities and dependencies between each step abstractly or concretely. Usually, a workflow can be represented as computational steps and the data that goes through these steps. Sometimes, the workflow composition process can be divided into two phases. First, a high-level workflow representation template is constructed, and then the actual data will fill into this template. These workflow templates can be stored and reused in other associated cases. I categorise the workflow composition mechanism in the following three types: textual, graphical and mechanism-based semantic models.

Textual Workflow Editing Many scientific workflow systems utilise a specific workflow language, such as Business Process Execution (BPEL) [42], Simple Conceptual Unified Flow Language (SCUFL) [43], Directed Acyclic Graph Manager (DAGMan) [44] and Directed Acyclic Graph in XML format (DAX) [45]. These workflow languages support text editor based composition with a specification. However, some workflow systems become extremely complex when the parallelisation is finished, where the algorithm is applied to several data sets through the entire workflow. Therefore, a high-level programming language (Python) based scripting tool is designed to describe the complex control and data flows. For instance, Pegasus [45] adopts DAX as a description language to generate workflow under a Java API and other types of scripting languages. Commonly, Pegasus is utilised to integrate with a user interface to provide interaction between workflow systems and users in terms of metadata querying. A workflow instance can be generated automatically with user input and then be executed based on DAGman.

Graphical Workflow Editing A graph-based workflow editor can simplify users' lives with intuitive graph editing. However, it is available in the domains where the workflow only has a small number of primary tasks, such as [46, 47]. As a result, many workflow systems like Kepler [48], Triana [49] and Vistrails [50] provide availability of composing nested workflow graphs. BPEL is a widely used workflow composition language integrating into many programming languages, such as Python and Java. For example, Eclipse BPEL Designer [10] employs primarily visual modelling for workflow composition. Two general types are considered to represent most of the workflow systems: task-based and service-based. Task-based systems focus on low-level workflow

[10]https://www.eclipse.org/bpel/

execution, such as resource management and fault-tolerance while service-based systems place emphasis on high-level user interaction to provide services management and composition. These two types of system can perform together; for instance, Kepler (task-based) works with Pegasus (service-based).

Semantic Workflow Editing Many scientists [51] focus on the development of fully automatic workflow generators with artificial intelligence techniques. This can be achieved through integrating semantic specifications of workflow components with standard features of correct workflows. Work like [52] utilises rich semantic descriptions of components and workflow templates expressed with regard to domain ontologies and constraints. Wings provide a workflow template editor for users to edit and compose workflow components with data flows, along with constraints specification. With information inputted by users, Wings generates a workflow instance including orchestration and data specifications. Since workflow representation and specification becomes declarative and expressive, these semantic workflow systems empower automation and assistance in workflow composition.

2.3.1.3 Scientific Workflow Scheduling

Workflow scheduling represents the process of resource selection for the executable workflow instance. In some cases, users perform these selection operations themselves. Nevertheless, more frequently, workflow systems can handle these processes. There are three main procedures in workflow scheduling: resource discovery, workflow mapping and workflow optimisation. I discuss such procedures in detail as following.

Resource Discovery A resource typically refers to a service which can complete one or more particular tasks in workflow systems. These services can be REST [53], WSDL-based Web Services [54], a computational agent, even a human. For example, a workflow step could be requiring a person to perform a particular task which can not be done by machines. In the web service context, a registry containing service descriptions is utilised to maintain existing services in workflow systems. When a new service is added, the registry may be updated within this new service. A search engine that can query the registry can be considered as a resource discovery tool.

Workflow Mapping As discussed, workflow systems perform workflow mapping for users in most cases. Kepler, Sedna, Taverna and VisTrails [50] rely on users to select and allocate resources and services used in workflows. Users need to select multiple alternatives for each workflow step to avoid undesired errors. A workflow engine, P-GRADE [55] based on Condor DAGMan and GEMLCA [56] extensions supports both task-based and service-based workflow applications. Similarly, Triana integrates with GAT (Grid Application Toolkit) [57] and GAP (Grid Application Prototype) [58] to offer task-based and service-based workflow mapping capability respectively. In a service-based workflow application, users can directly invoke services, or they can compose a workflow then map it to distributed services via a pipeline script. On the other hand, Triana can send user-defined tasks to available resources in task-based workflows. Karajan [59] provides dynamic binding of tasks to resources by allowing users to specify tasks at an abstract or concrete level, then mapping such specified tasks to a single resource. In the BPEL based workflow systems, the resource selection and workflow mapping processes will always be explicit as abstract Web Service Definition Language (WSDL) descriptions.

Workflow Optimisation When performing workflow optimisation, a composed scientific workflow can be improved in terms of execution cost, work efficiency, quality of service, etc. Pegasus provides a user interface with a user-defined scheduler including four standard scheduling algorithms: HEFT [60], min-min, round-robin and random. With such scheduling algorithms, Pegasus can schedule workflows depending on the quality of information about execution time and data accessibility. Meanwhile, Pegasus can offer four types of workflow optimisation: task clustering, data reuse, data cleanup and partitioning [45]. Another task-based workflow system, called the Askalon system, can perform resource mapping along with resource provision during execution. This system has a performance analysis and prediction component which estimates the data transfer time and the workflow tasks execution time.

2.3.1.4 Scientific Workflow Execution

When a workflow template is designed, a workflow execution engine or system is needed to perform and execute such workflows. I classify workflow execution in the following two ways: execution model and fault tolerance mechanism.

Execution Model Workflow systems generally handle services or tasks. In various execution scenarios, workflow execution follows different models. For example, Pegasus can schedule workflows in a range of target resources managed by PBS [61], LSF [62], Condor [63] and individual machines. Pegasus adopts DAGMan workflow engine as an execution tool and GSI [64] as remote resources accessing the tool. In the meantime, Askalon adopts the Unified Modelling Language (UML) standard as composition and modelling technique, and supports XML-based Abstract Grid Workflow Language (AGWL). The Java-based workflow framework, Karajan provides DAG-based hierarchical workflow structures. In this system, workflows can be visualised and tracked by a workflow scheduler and modified at runtime. Triana integrates with the GridLab GAT to support task level workflow execution. Besides, it binds to Web, WS-RF and P2P services to support service-level workflow execution as well.

Fault Tolerance In a different level of workflow, the fault tolerance mechanism can be various. Some operating-system-level fault tolerance mechanism can be used to save workflow execution time and recover system failure. Moreover, application-level fault tolerance mechanisms, such as check-pointing, can support workflow execution migration. The Cactus worm [65] adopts this mechanism to solve task-level workflow execution failure. On the other hand, operations to perform fault-tolerance in a service-level workflow can restart services in a time interval or mitigate to another alternative service. For example, Kepler integrating with a "smart re-run" extension can avoid unnecessary re-computations. Similarly, Triana informs users with error messages when failure has occurred. DAGMan supports a tasks re-submitting feature with a batch system. Apart from these mechanisms, to monitor runtime information externally is another choice. For example, when executing a workflow, an external monitor agent can detect failures happening and bring failed tasks to a substitute

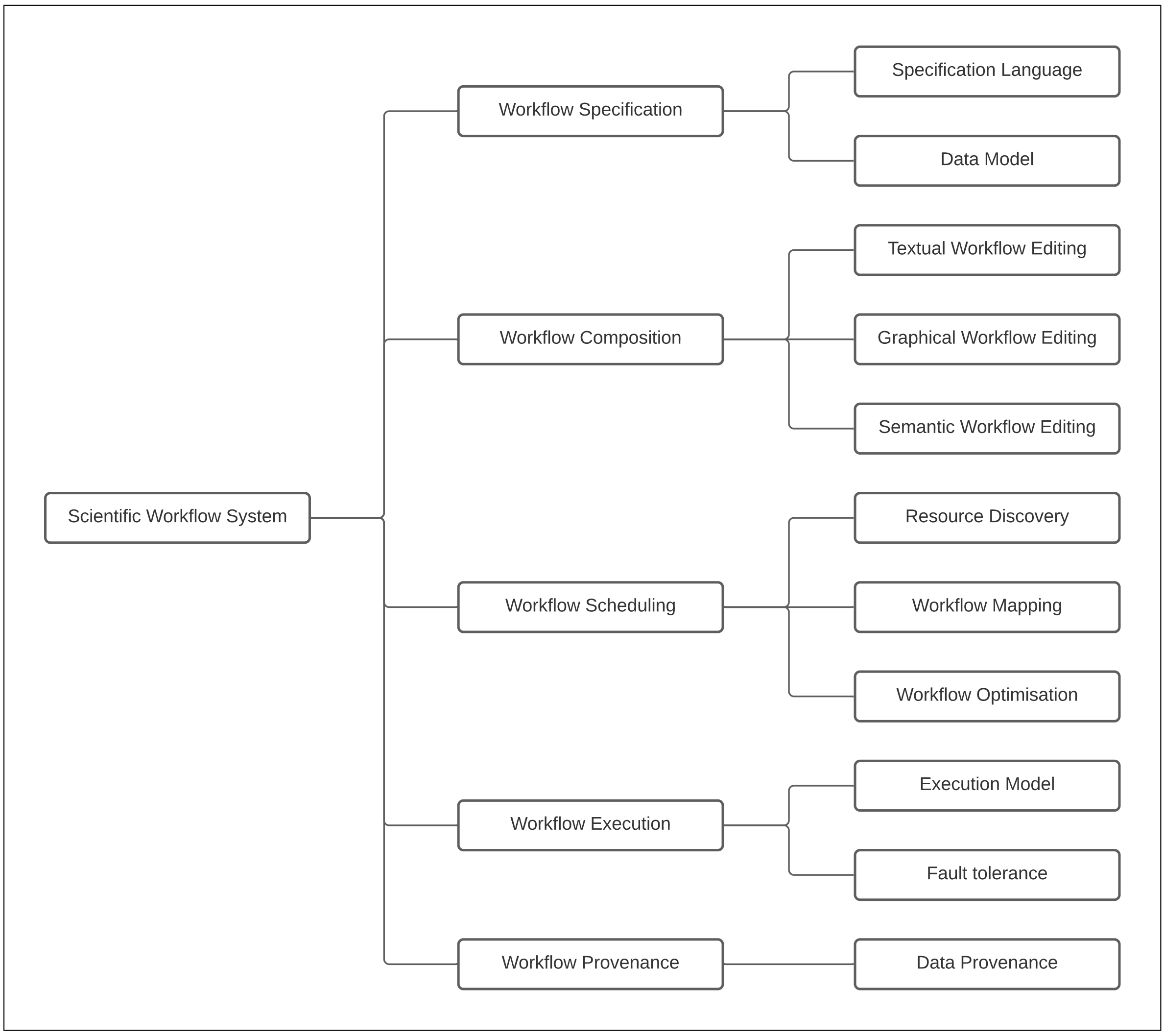


Figure 2.4: Taxonomy of Scientific Workflow System

resource.

2.3.1.5 Scientific Workflow Provenance

Workflow provenance holds a record of a workflow data object from creation. To hold such records is important because users can reproduce the workflow result and analysis workflow in terms of particular aims.

The increase of workflow provenance challenge comes from various provenance representations from different workflow systems even the same workflow is produced. In other words, Some systems manage provenance information with internal data structures; others use external services to do so. For example, Triana records provenance information locally in an internal format. In this record, the execution history and parameters of the data sets are easy to access and understand.

The VisTrails system [66] can track the provenance of workflows with the capability of workflow management. Users can explore any operations on a workflow via a graphical user interface. With this kind of workflow provenance, scientists can quickly revise and analysis the workflow across domains.

In summary, the taxonomy of scientific workflow system is illustrated in Figure 2.4.

2.3.2 Internet of Things Workflow

In the past decades, IoT devices have generated a massive amount of raw data for processing purposes. Cloud and Edge are two major paradigms in the distributed computation environment. Despite cloud datacenter having massive computational capability, transferring all data generated from IoT devices to the centralised cloud datacenter for processing is unnecessary. This operation may increase end-to-end latency and response time. It also raises network resource wastage and energy consumption rate. An edge datacenter located close to IoT devices with lightweight computational power is proposed to overcome these disadvantages. The critical benefit of an edge datacenter is to perform preprocessing with the raw data captured from IoT devices in order to reduce the data size to transfer to centralised cloud datacenter. The end-to-end latency is also decreased because edge devices always deploy locally. However, many edge nodes have computational capability limitation to handle large scale data processing tasks. Thus, neither cloud nor edge datacenters can manage various data processing tasks efficiently. It raises a new challenge to formulate and pipeline an IoT application with both edge and cloud resources integrated. IoT workflow application is proposed to address the above challenge in order to utilise cloud and edge resource capability to meet the quality of service or other service level constraints.

An IoT workflow similar to a scientific workflow is represented as a DAG with dependencies [67–69]. The workflow refers to a set of scientific tasks or engineering problems. Every workflow application requires massive computational power to execute distributed tasks over resources in an expected time [70]. Because of the IoT application diversity, workflow composition and orchestration, which refers to the scheduling of workflow, leads to specific challenges and opportunities.

2.3.2.1 IoT Workflow Composition and Orchestration

In order to orchestrate relevant resources to handle corresponding tasks, for example, large scale data processing tasks need heavy computational power in the cloud, an intelligent IoT workflow composition mechanism needs to be developed. Meanwhile, meeting a range of quality of service constraints, such as latency, cost, etc., is another essential characteristic of IoT workflow composition and orchestration.

Cloud-based workflow composition and orchestration have been studied comprehensively [71–74]. However, such studies focus on centralised cloud environment workflow analysis which can hardly cover current cloud-edge associated IoT deployment environments.

Objectives of IoT Workflow Composition and Orchestration The workflow composition and orchestration strategy in IoT differs from traditional scientific workflow composition because it needs to address several research challenges for enabling efficient and effective operations. When deploying an IoT application which contains many different data processing tasks, the computational time of each task depends on the data size and task type. The primary objectives [75–77] to compose and orchestrate IoT workflow can be summarised as follows:

- **latency**: Discovering the closest resource to execute an IoT workflow task is a primary challenge in a federated cloud-edge environment [78]. The latency describes the time interval between an output from the previous step and input of the next step. It relies on geo-distance and network quality.

- **Energy**: Designing a workflow which can reduce energy consumption is a challenge in IoT [79]. The energy consumption of a resource varies leaning on the processors and memory size of the resource and also the resource utilisation. As a result, executing IoT workflow steps on edge nodes can control the total energy consumption, while waste and temperature controlling is affected by resources energy consumption.

- **Resource Utilisation**: Resource utilisation refers to the number of resources that are utilised by the tasks within a time interval [80]. Loads are assigned on remaining resource capability. If the remaining load of local resources is less than the required load, the workflow steps need to be executed locally. Alternatively, such loads should be allocated to centralised cloud resources for further execution. Composing and orchestrating an IoT workflow to enable high resource utilisation is a challenge.

2.3.3 IoT Workflow Composition and Orchestration Tools: State of the Art

Workflow composition tools in IoT offer various features, such as workflow definition, workflow design, workflow execution, etc. These tools and platforms can significantly improve scientists' work in terms of IoT related workflow deployment and orchestration. In this section, I describe nine popular IoT workflow tools and platforms and analyse the features and advantages of them below.

Node-RED Node-RED [11] is a flow-based event-driven pipeline application that consists of a Node.js runtime with a web browser to access the workflow editor. In Node-RED, users can pipeline and model workflows with provided "nodes" as a network of black-boxes of an application's behaviour. Each node is well-defined to perform a specific purpose. In IoT workflow applications, Node-RED can provide a platform which allows users to represent their workflows visually and debug workflow steps in runtime. Workflows are specified as a JSON metadata which contains node information and dependencies. Such JSON files are easy to share with other IoT workflow application developers to import and reuse in other scenarios. Besides, Node-RED is a lightweight workflow system which can be deployed on locations with low computational power such as Raspberry Pi. This feature increases the capability of workflow composition and orchestration. However, scalability is a big challenge for Node-RED in the current version because the software is a Node.js runtime environment and when workloads increase, it has the potential to fail.

[11]https://nodered.org/

Airflow Apache Airflow [12] is an open-source workflow management platform to programmatically author, schedule and monitor workflows. In Airflow, all workflows are represented in code which is easy to maintain, test and collaborate. Users can visualise workflow pipelines as directed acyclic graphs (DAGs) of tasks which are executed further by an Airflow scheduler following the specified dependencies. Airflow is a pure python-based software supporting standard python features, including data time formats for scheduling and loops to generate tasks dynamically. Moreover, Airflow provides a range of plug-and-play operators to perform typical cloud platform execution, such as Amazon Web Services, Azure, Google Cloud Platform, etc. Scalability is also enabled by applying a message queue to orchestrate an arbitrary number of workers. However, Airflow still has some weaknesses; one is that version control is missing. If users delete a task from the DAG, then associated metadata will be lost. Another disadvantage is about data sharing between tasks. Recently, data is sharing between tasks benefits, data reuse and data aggregation processes. There is no easy way to share data between atomic tasks in Airflow. Users can only restart ETL (extract, transform, load) whenever any part of it fails.

Argo Workflows Argo Workflows [13] is an open-source container-native workflow engine for parallel tasks orchestration based on Kubernetes infrastructure environments. It is implemented as a Kubernetes Custom Resource Definition (CRD) to benefit workflow management using kubectl and natively integrate with many well-defined Kubernetes services, like secrets, volumes and RBAC (Role-based access control). Argo is easy to deploy on many resources in a short period and offers parameter substitution, artefacts, loops, fixtures and recursive workflows features. When performing Argo as a workflow engine, users can design their DAGs in a provided workflow specification template. Meanwhile, several properties such as workflow type, container information and resource information are defined in each pod of a workflow. After workflow script is well designed, Argo continuous delivery software can be utilised for declarative workflow deployment. Since the workflow template is written in YAML format script and tasks are defined in a containerised environment, reusability and extensibility are easy

[12]https://airflow.apache.org

[13]https://argoproj.github.io/projects/argo

to achieve. However, continuous integration is not supported in Argo, which means test and build steps are lacking in Argo workflows.

CDS Continuous Delivery Service [14] is an open-source platform to offer enterprise-grade continuous delivery and DevOps automation services written in Go language. In CDS, they conceptualise workflow applications as two parts, pipelines and workflows. Pipeline refers to structured, sequential stages containing one or more concurrent tasks, while workflow represents a chain of pipelines with conditional branching. They consider workflows at high-level, including not only small jobs and tasks but also all operations to perform an application from scratch. The benefit of these divided concepts brings fault-tolerance to CDS because users only need to replace or modify a pipeline when a particular step fails instead of debugging the entire workflow. CDS contains many user features like git repository configuration, native GitHub/GitLab integration, secure remote caching, etc.

Cadence Cadence [15] is a distributed, scalable, durable and highly available orchestration engine to execute asynchronous long-running business logic in a scalable and resilient way. In Cadence, fault-oblivious stateful workflow code is the primary abstraction. The form of the workflow code, including associated local variables and threads it creates, is immune to process and Cadence service failures. The workflow encapsulates states, processing threads, durable timers and event handlers to offer state recovery and determinism. Activities representing functions or object methods are orchestrated by workflow code to communicate with external API directly. Such activities are invoked asynchronously through task lists, storing a queue of available activity tasks. Cadence supports Java and Go application as a client application and uses a rich command-line interface to perform essential interaction. However, a complex Java/Go deployment environment declines lightweight computational resources in the edge.

[14]https://ovh.github.io/cds/docs/
[15]https://cadenceworkflow.io

n8n n8n [16] is an extendable workflow automation tool with a fair-code distribution model allowing users to add customised functions, logic and apps. With n8n, metadata can be moved and transformed between apps and databases without integrating with particular APIs or troubleshooting CORS (cross-origin resource sharing) error. A user-friendly user interface is provided for workflow designing within javascript functions and conditional logic. A well-defined workflow consists of nodes which retrieve data and execute functions in sequence. These nodes can be any services, such as cloud services and any functions with Node.js runtime. A trigger node is designed to start a workflow and supply the initial data. Reusability and extensibility rely on a JSON based script editing and sharing mechanism. Some important runtime information is stored within the workflow script, including time stamp, execution error data, execution URL, etc.

Luigi Luigi [17] is a python based pipeline tool for batch tasks. This software is designed to solve long-running batching processes in data science domains, such as Hadoop, Spark, Hive, etc. It hides the complexity of workflow management to provide focus to data scientists on their tasks and dependencies. A web-based user interface is employed for workflow pipeline and task searching and filtering. Some features, like failure handling, task tracking and event handling, make Luigi more reliable for batch data processing. Integrating with such batch data processing platforms (Hadoop, Spark) is a crucial advantage for users to execute MapReduce jobs and Spark jobs. To avoid data pipeline crashing, Luigi provides file system abstractions for HDFS (Hadoop Distributed File System) as standard task templates for usability. However, due to the batch processing oriented design, Luigi is less useful in real-time workflow pipeline and continuous delivery. Additionally, distribution of execution is not supported in Luigi when worker nodes get overloaded, which leads to failure.

Prefect Prefect [18] is a workflow management system powered by the open-source Prefect Core workflow engine and designed for modern infrastructures. Users only need to focus on tasks definition and flows composition. The Prefect Core workflow engine

[16]https://n8n.io
[17]https://luigi.readthedocs.io/en/stable/index.html
[18]https://www.prefect.io

allows semantics like retries, logging, dynamic mapping, caching, failure notification to formulate workflow execution. In a Prefect workflow, Tasks represent discrete actions which execute user-defined functions, while flows refer to entire workflows or applications by illustrating the dependencies between tasks. Prefect supports a wide range of common deployment solutions involving AWS, Azure, GCP, Docker, Kubernetes, etc. and many task libraries like MySQL, Python, etc. Nevertheless, Prefect is friendly for edge devices without command-line accessibility.

Temporal Temporal [19] is a microservice orchestration platform which enables scalable applications development with productivity and reliability. Units of application logic, workflows are executed on a Temporal server in a resilient manner that handles intermittent failures and retries failed operations automatically. Temporal currently provide SDKs for Go and Java programming languages with pseudocode-like workflow representation. Workflow operations like pause, resume and replay in the provided SDKs enable state management, queueing, resilience and other safety features. Any Temporal environment includes a backend service, workers and a data repository. Workers running end-users' functions can connect to the Temporal service, which offers to schedule, queueing and state management via a gPRC (remote procedure call) protocol. An external data repository, such as MySQL or Cassandra can record and store data from Temporal service side for further reusing. Temporal can be considered as an extended version of Cadence with improvement in communication protocol changing and configuration simplifying.

In summary, the comparison of these workflow compositions and orchestration tools in terms of the deployment environment, scalability, extensibility, etc., is shown in Table 2.3

[19]https://temporal.io

Table 2.3: IoT Workflow Composition and Orchestration Tools Comparison

Platforms	DAG	Access Mechanism	Workflow Represen -tation	Deployment Environment	APIs and SDKs	Fault Tolerance	Scalability	Extensibility	Reusability
Node-RED	Yes	Web-based User Interface (Web UI)	JSON	Cloud, Edge, Docker, Android	Node.js API, HTTP API, Javascript base Storage API, jQuery widgets API	Yes	Yes	Yes	Yes
Airflow	Yes	Web UI, Command Line Interface (CLI)	Python	Cloud, Edge, Docker	Python API, REST API	Yes	Yes (modular architecture)	Yes	Yes (Workflow as Code)
Argo Workflows	Yes	CLI, Web UI for CD only	YAML	Cloud, Edge, Docker	Swagger API, REST API	Yes	Yes (horizontally scale the worker not controller)	Yes	Yes
CDS	Yes	CLI, Web UI	YAML	Cloud, Edge, Docker	REST API, Golang SDK, Rust SDK	Yes	Yes	Yes	Yes
Cadence	No	CLI, Web UI	YAML	Cloud, Edge, Docker	Java API, Go API	Yes	Yes	No	Yes
n8n	Yes	CLI only for workflow execution, Web UI	JSON	Cloud, Edge, Docker	REST API,	Yes (error trigger)	Yes	Yes	Yes
Luigi	Yes	Web UI	Python	Cloud, Edge	Python SDK	Yes	No	Yes	Yes
Prefect	Yes	Web UI, CLI	Python	Cloud, Edge, Docker	Python SDK	Yes	Yes	Yes	Yes
Temporal	No	CLI, Web UI	Java/Go	Cloud, Edge, Docker	Java SDK, Golang SDK	Yes	Yes	Yes	Yes

2.4 Conclusion

In this chapter, I have discussed the background information of associated topics including IoT, scientific workflow systems, configuration management, deployment automation and IoT workflow composition. From the literature review, I have illustrated recent academic research and analysed drawbacks on such topics. In addition, popular configuration management and IoT workflow composition tools are discussed and evaluated in terms of several critical dimensions. In summary, regarding the above discussion, I highlighted research challenges concerning IoT workflow composition and deployment automation.

3

IoT-CANE: A Unified Knowledge Management System for Data-Centric Internet of Things Application Systems

Contents

Summary

In this chapter, I present IoT-CANE (Context Aware recommendatioN systEm) as a high-level unified IoT resource configuration recommendation system to overcome resource heterogeneity in cloud and edge environments. IoT-CANE embodies a unified conceptual model capturing configuration, constraint and infrastructure features of Cloud/Edge together with IoT devices. The success of IoT-CANE is evaluated through an end-user case study.

3.1 Introduction

The Internet of things (IoT) commonly refers to the idea of things (devices) that are smart, locatable, easily readable and recognisable and controllable using the Internet [1–3]. In IoT, things and resources (edge and/or cloud), combine as services that form the basis for many IoT applications e.g. smart healthcare, smart homes, smart buildings, smart manufacturing, smart agriculture, smart traffic, etc. In such applications, real-time users can easily discover, query and operate different IoT entities. Numerous physical objects provide these kinds of services using IoT technologies. These objects are not only physical objects, such as physical sensors and actuators, but also virtual objects, like social media (Facebook, Twitter, etc.,). The heterogeneous large-scale data from such physical and virtual objects raises a challenge of unified resource configuration knowledge representation and high performance data processing [8]. Therefore, we need an incorporation of the digital world and the physical world in IoT ecosystems; that is, in order to allow interoperability, it is important to define the services supplied by these physical and virtual objects in a homogeneous way [9]. More specifically, the requirement of developing a novel and unified conceptual model to represent the knowledge and configuration information of each entity in IoT field is necessary.

Consider a scenario of a new IoT application manager that would like to create for a smart building application without high budget. The manager may purchase IoT devices from multiple manufacturers with a low price, such as temperature sensors, motion sensors, humidity sensors, Raspberry Pi, gateway, etc. We assume this manager does not have enough knowledge of Cloud and Edge resource configuration management and deployment. Because of the heterogeneous data type, diverse hardware and software knowledge in different IoT devices and Cloud/Edge resources, the manager needs to find some IoT application solution providers to offer IoT resource configuration management and deployment solutions. However, the problem is that the majority of IoT application solution providers that can offer different IoT application solutions to the user, are expensive. Moreover, IoT application solution providers can only provide existing services within their service list with the IoT devices supported by their software. Besides, IoT devices produced by different manufacturers may have different APIs, which increases the difficulty of service deployment from a single IoT application solution provider. For example, currently, there are more than 325 available APIs for Internet of Things programming from the ProgrammableWeb website [1]. Therefore diverse programming APIs and multiple IoT devices bring new challenges in configuration management and deployment of IoT applications.

In another scenario, a householder purchases a new smart camera to enhance his smart home application with an intrusion detection service. He may need to connect this new smart camera to his smart home environment, for example, by establishing a connection between the camera and a gateway to collect graphic data. After that, another node in the smart home application captures this graphic data and detects any intrusion via graphic data processing. The householder may have already implemented the smart home with the help of professionals. But it may not be affordable or practical to seek help every time when he purchases new IoT devices. In this case, discovering and adding a new device or a new service into an existing IoT application becomes a challenge for the beginners. Also, the new data type from smart camera, such as format of photos and videos are new for his smart home application. Meanwhile, one or multiple software which are eligible to handle with such data format are necessary to be aggregated into this smart home application. Therefore, many IoT application users will meet diverse problems in IoT resource configuration management and deployment due to rapid IoT development.

To address such growing challenges, we present a high-level, simplified context-aware

[1]https://www.programmableweb.com/category/internet-things/api

IoT resource configuration recommender system (IoT-CANE) to address the aforementioned specified problems. The core idea in our IoT recommender system is to first, capture the IoT resource configuration knowledge using a declarative language, and implement a recommender system associated with a relational data-model. Execution in the IoT-CANE uses transactional procedures and applies SQL-based relational semantics for different IoT resource configurations operations, such as querying, inserting and deleting. Also, the IoT recommender system is designed to facilitate increased acquisition of knowledge.

The key contributions of this chapter are as follows:

- We propose a unified conceptual model which captures the resource configurations in IoT environments. The model incorporates various available infrastructure features for IoT devices, Edge and Cloud.
- We design a support recommender system for the recommendation of resource configuration in IoT using SQL-based relational semantics and procedures.
- We develop an incremental method to facilitate the knowledge acquisition in IoT resource configuration knowledge base.
- We design a service interface that converts simple context information captured from users to optimal IoT resource configurations to map users' requirements.

The rest of this chapter is organised as follows. Related work about resource configuration management issues in IoT, IoT conceptual model and context-aware recommender system is presented in Section 5.2. A detailed description of the conceptual model and system architecture is presented in Section 5.3. Performed techniques and implementation of the proposed recommender system are presented in Section 3.4 and Section 3.5 respectively. Evaluation of a use case study is provided in Section 5.5 before the conclusion and future work in Section 5.6.

3.2 Related Work

3.2.1 Multi-layer Resources Configuration Management Issues in IoT

The recent trend in composing Cloud applications is driven by connecting heterogeneous services deployed across multiple datacenter [81]. Such a distributed deployment aids in improving IoT application reliability and performance within Edge computing environments. Ensuring high levels of dependability for such IoT data transformation tasks composed by a multitude of systems is a considerable issue. Different frameworks for describing and deploying Edge and Cloud resources are proposed in academia and industry. Multiple Cloud providers such as CA AppLogic [2] and AWS OpsWorks [3] allow description and deployment of complete Cloud application stack. They offer resource representations that are specific to a particular provider only. In Edge computing, Docker [4] provides deployment and configuration management solutions for Edge devices based on container techniques. However, when it comes to IoT, in addition to the Cloud and Edge layers, IoT resources configuration management should also consider the physical devices which are deployed widely in most IoT applications. In IoT, all resources from multiple layers need to be considered in a single application which leads to much more complicated scenarios.

3.2.2 Conceptual Model in IoT

The Semantic Sensor Networks ontology presents a high-level conceptual model to describe physical devices, their capabilities and the associated properties in the semantic sensor networks within the IoT area. Authors in [82], provide a description of IoT application elements, and a data model, and also capture the relationships among various data provider and descriptor cells. They also illustrate how their models can be associated with each other and can be related to different domain knowledge. The IoT-A project has also described services, entities and resources as basic concepts in the IoT domain [83]. The entity in their model is the main core of interactions by

[2]https://support.ca.com/us/product-information/ca-applogic.html
[3]https://aws.amazon.com/opsworks/
[4]https://www.docker.com/

Table 3.1: Comparison of Related Work

Parameter	Related Work							IoT-CANE
	[85]	[82]	[83]	[84]	[86]	[87]	[88]	
IoT device conceptual model	✓	✓	✓	✓	✗	✗	✗	✓
EDC,CDC conceptual model	✗	✗	✗	✗	✗	✓	✓	✓
IoT ontology modelling	✓	✗	✓	✓	✗	✗	✗	✓
knowledge recommendation	✗	✗	✗	✗	✓	✓	✓	✓
incremental knowledge base	✗	✗	✗	✗	✗	✓	✗	✓

people and software agents. As they proposed, an IoT service reveals the functionality of a resource which is hosted on devices that offer physical access to the entity. In [84], the authors present the design of a completed and lightweight semantic description model for knowledge representation in the IoT field. The widely applied rules in knowledge engineering and ontology modelling are considered in their design. However, their model only considers the physical world of the IoT domain and how to abstract the physical devices. They do not consider Edge and Cloud components, which are extremely important in unified knowledge representation of diverse IoT applications.

3.2.3 Context-aware Recommender Systems

The context notion has outstandingly matured from what it was proposed in [89]. Currently, there are different types of contextual information, which are mainly categorised into three different classes: physical context, user context, and appliance context based on the adopted views e.g. user or application perspective. The meaning of context depends on the area and structure of the considered application, which makes it appropriate to provide a proper definition of the concept. Possible recommender system interpretation for applications like e-commerce and different systems are provided in [90]. Location is a common contextual piece of information in recommender systems, however, it does not represent the users' geo-location all the time, e.g. for the social tag-based joint filtering method given in the context of smart TV applications [86]. This is a context-aware approach where the information of both user and device contexts are considered. In this case, the recommendations are calculated only using users' preferences and the recommendations are re-ranked appropriately. In [87], the authors presented a framework to provide declarative context driven knowledge

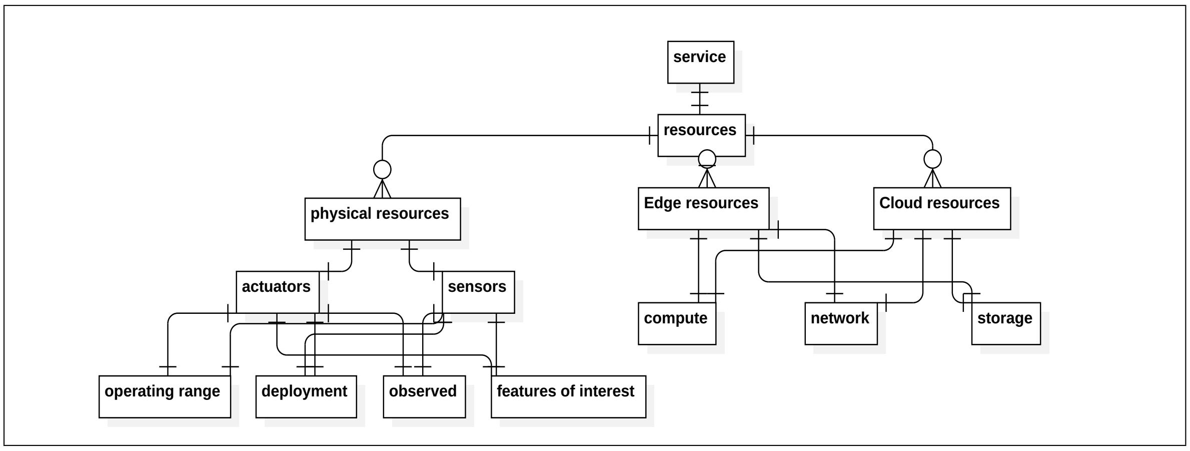


Figure 3.1: The conceptual model of IoT resources

recommendations for federated Cloud resources configuration. In this framework, they give a recommendation of configuration knowledge artefacts based on a given context. However, these frameworks provide a context-driven recommendation in Cloud computing while in our approach, context-aware recommendations are given in the IoT area which is more complicated. In [88], the authors proposed an ontology-based infrastructure selection system based on the real-time QoS requirements and utilised analytic hierarchy process method to facilitate multi-criteria decision-making. However, their infrastructure selection system is designed only for Cloud resources selection, IoT resource selection is still a significant research problem. Such related work can be summarised in the following comparison Table 3.1.

3.3 Conceptual Model and System Architecture

In this section, we present our conceptual infrastructure model for IoT, the IoT-CANE system architecture.

3.3.1 Conceptual Model

An IoT framework can easily take advantage of various models which provide different concepts and abstractions for the components and their respective attributes. The main concepts and abstractions underling the IoT infrastructure and describing relationships are presented in this section. One main notion of our research is the representation of unified knowledge for IoT resource configurations. A unified hierarchical representation data-model is presented using the entity-relationship modelling (ER model), shown in Figure 4.1. The physical resources part of our model is designed based on Semantic Sensor Network (SSN) Ontology [5]. The Semantic Sensor Network (SSN) ontology is an ontology which depicts various sensors and their observations, the related procedures, the samples used to do so, the considered features of interest, and the observed attributes, as well as actuators. The designing of SSN follows a two-dimensional modularisation by implementing a lightweight but self-contained core ontology called SOSA (Sensor, Observation, Sample, and Actuator) for its primary classes and attributes. In SSN ontology, sensors and related concepts, are described without domain concepts, e.g. time, locations, etc. In our conceptual model, we considered these domain concepts because our model is suited for some specific domains and further OWL imports are not necessary for our conceptual model. Moreover, Edge and Cloud resources are also considered to complete the conceptual infrastructure model for IoT as these resources are also configurable and describable.

In Fig. 4.1, the widget called resources in IoT conceptual infrastructure model consists of three main entities, physical resources, Edge resources, and Cloud resources. These three entities represent the abstraction of physical devices (e.g., sensors, actuators, etc.), Edge devices (e.g., gateways, routers, etc.) and Cloud infrastructure (Datacenter) respectively. Next, we discuss these entities in detail.

- *service*: the service entity represents the domain information of IoT services, each of them provides an organised and standardised interface that offers all the necessary functionalities for interaction with resources and the related processes. The services expose the functionality by accessing multiple resources. The service entity consists of a service profile, service grounding and service model subclasses. The type of service can be categorised as OWL-S, USDL, WSML, WADL, SOAP and etc. Service profile describes the semantic description and textual information of a service. Service grounding provides the attributes for service interaction and access, such as endpoint address, input and output. Service model depicts the operations and outcomes inside each service.

[5]https://www.w3.org/2005/Incubator/ssn/ssnx/ssn

- *resource*: resource class is the high-level class of physical, Edge and Cloud. It contains resources, name, type, access interface, description and etc. This class provides a general description of IoT resources which expose IoT services.

- *physical resource*: physical resources class describes the attributes of sensors and actuators in the conceptual model. Both actuator and sensor have their specific type, location name, geolocation, latitude, longitude, availability and etc. The type of sensors and actuators can be easily abstracted in three categories, physical (temperature sensor, locker), virtual (Facebook) and smart (smartphone, smart camera). The smart here indicates the physical resource which has the computational capacity so that it can be programmed.

- *operating rage*: operating range indicates the conditions in which a sensor/actuator is expected to operate. It contains resolution, response time, measurement range, precision, latency and accuracy.

- *deployment*: deployment class describes the deployment of a or several specific sensor/actuator for a certain purpose. For instance, a motion sensor can be deployed on the corner of a room.

- *observed*: observed indicates a sensor/actuator is observed in a particular method. It consist of accuracy, observation, observed result, result time and sampling. The observed class links sensor/actuator and feature of interest.

- *feature of interest*: the feature of interest describes the object which associate with sensor/actuator and estimate observation. It has role, property and object attributes. For example, when you measure the depth of a river, the depth is the observed property, 50 meters may be the observation result and the river is the feature of interest.

- *Edge resource*: Edge resource indicates the resources deploying in the Edge leverage computational capacity to improve latency, privacy and security. It contains performance, location name, availability, edge resource type, attached IoT device, battery capacity, current battery and etc. Edge resource capture data from sensor and send command to actuator.

- *Cloud resource*: Cloud resource describe the cloud infrastructure deploying in a variety of cloud service providers. It consists of performance, location name, geolocation, availability and connected edge devices. The type of cloud resource can be virtual machine, container, virtual storage and etc. Both Edge resource and Cloud resource have subclass as compute, network and storage.

- *compute*: it depicts the computational capacity of each Edge or Cloud resource. Hypervisor, CPU number, CPU cores, RAM, operating system are the main attributes of the compute class. In a IoT application, resource computational capacity may influent the decision of resource configuration selection. For example, if a user attempts to deploy a Hadoop cluster to process a large volume data, a high performance virtual machine may be the better choice than a limited computation raspberry pi.

- *network*: the network describes all the network connection between each two entities, such as the communication between Edge and Cloud, the data transformation from sensor to Edge, and etc. It contains response time, network bandwidth, uplink bandwidth, downlink bandwidth, latency and etc. Network configuration manages all the entity communication and data transformation tasks which require the functionality of stability and fault-tolerance.

- *storage*: storage class providing the storage capacity of Edge and Cloud resource consists of storage capability, storage type, storage bandwidth and etc.

All these attributes contain main functional configuration of every single Edge/Cloud resource attached into a proposed IoT service. Based on these properties, the Edge/Cloud node of an IoT application can be easily deployed.

To explain the conceptual infrastructure model clearly, we present a partial description of infrastructure components for a smart building scenario using the proposed conceptual model.

As shown in Table 3.2, the sensor class contains five attributes – sensorType, locationName, latitude, longitude and availability – to describe resource type, geographical area and running time availability respectively; the actuator class contains five

Table 3.2: Partial infrastructure components description and instance of smart building in IoT data model

Class	**Attribute**	**Type**	**Description**	**Instance in Smart Building**
resource	hasName	String	Name of the resource	Smart temperature sensor
	resourceType	Resource type	Resource type, such as physical resource, edge resource and cloud resource	Physical resource
	accessInterface	Interface type	Interface type, such as REST, SOAP and XML-RPC	REST API
	hasDescription	String	Description of the resource	Intelligently measure the temperature of the specific area
	hasTag	String	Any tags of the resource	Smart temperature
actuator	actuatorType	Actuator type	Actuator type, such be physical (locker), virtual (social media) and smart actuator	physical
	locationName	String	Geographical area in which the resource is located	Newcastle
	latitude	Float	Latitude of the actuator	54.9783 N
	longitude	Float	Longitude of the actuator	1.6178 W
	availability	Boolean	Actuator availability	available
sensor	sensorType	Sensor type	Sensor type, such as physical (temperature), virtual (social media) and smart sensor	physical
	locationName	String	Geographical area in which the resource is located	Newcastle
	latitude	Float	Latitude of the sensor	54.9783 N
	longitude	Float	Longitude of the sensor	1.6178 W
	availability	Boolean	Sensor availability	available
operating range	resolution	String	The smallest difference in the value of an observable property being observed that would result in perceptible different values of observation results	-
	responseTime	String	The time between a change in the value of an observed sensor	1.8-60 seconds
	measurement Range	String	A set of values that the sensor can return as the result of an observation	-55 to 150 C
	precision	String	As a sensor: the closeness of agreement between replicated observations on an unchanged or similar quality value; As an actuator: the closeness of agreement between replicated actuation of an unchanged or similar command	0.36 C (max)
	latency	String	The time interval between a command for an observation and the sensor providing a result	500ms (max)
	accuracy	String	The closeness of agreement between the result of observation (command of an actuation) and the true value of the observed	0.1 C

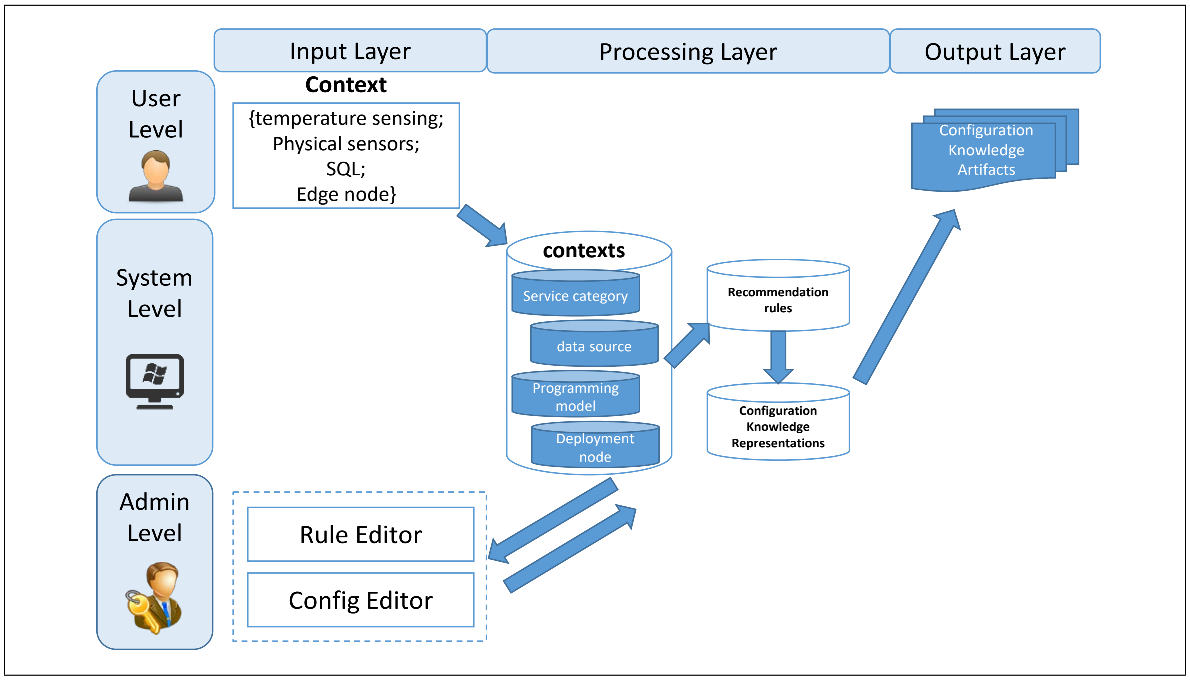


Figure 3.2: The system architecture of IoT-CANE

attributes as well; the operating range class contains resolution, responseTime, measurementRange, precision, latency and accuracy attributes; in each line, we presented a short description to explain the meaning of attribute. Additionally, in the last column of the proposed table, we present an instance in smart building to illustrate each attribute in detail. For example, we choose the Urban Science Building (USB) as a smart building application. USB is a smart building located in Newcastle upon Tyne. There are more than 4000 sensors throughout the building providing high resolution data regarding environmental, mechanical and electrical performance. In USB, the instance of the sensor can be a temperature sensor in the 3rd floor. Then the resource name is 'smart temperature sensor', the accessInterface is 'RESTful API', the locationName is 'Newcastle USB', availability is 'available' in this case.

3.3.2 System Architecture

This context-aware recommendation system is intended to suggest CKAs to multiple layers required by users during IoT resource configuration management processes (such as IoT resource deployment and configuration parameter modification) in an automatic way. CKA represents configuration knowledge artifact which is a set of configuration details of a concrete service. As shown in Figure 3.2 Recommended suggestions are generated based on the users' context (e.g. service category, data source and programming model). This context represents an individualised IoT data transformation task or an IoT service requirement. All the necessary instructions and information required to satisfy the context description for resource configuration of Edge or Cloud is included in recommended CKR. CKR represents configuration knowledge representation which consists various configurations storing in the database. Recommendations can be derived using CKAs, e.g. bundled virtual appliances and runnable deployment texts, using information from similar past contexts. Recommended CKAs can be accepted unchanged or modified according to users' specific requirements. Instead, users can generate a new CKA if they refuse the previous recommendation. After any new CKA defined by the user, the recommender system converts those modifications into recommendation rules and saves them to make them available for any future recommendation. Meanwhile, recommended CKAs are input to a Docker deployment engine to provide detailed configurations. The detailed discussion of IoT-CANE system will be presented in section 3.5.

3.4 Recommendation System Technique

In the IoT-CANE, we adopted a rule-based recommendation method to generate context-aware configuration recommendations. Meanwhile, Ripple Down Rules are employed to facilitate knowledge acquisition in configuration knowledge base. Detailed techniques are discussed in the section given below.

3.4.1 Recommendation Rule

In IoT-CANE, an IoT resource configuration knowledge base (CKB) is maintained by this recommender system that stores contextual information, CKRs and CKAs. The configuration knowledge base is a database which stores all configuration knowledge. An association is maintained between the items in the CKB by the recommendation rules as shown in the Figure 3.3. Recommendations include two components, named contexts and conclusions.

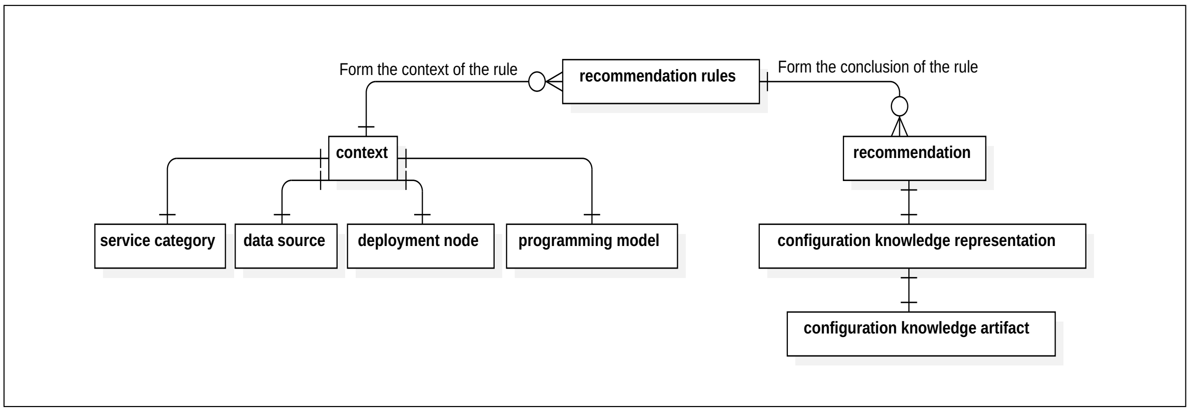

Form the context of the rule
recommendation rules
Form the conclusion of the rule
context
recommendation
service category
data source
deployment node
programming model
configuration knowledge representation
configuration knowledge artifact

Figure 3.3: ER diagram of recommendation rules

Contexts The left-hand side of the ER diagram contains the context information. The recommender system maintains "contexts" data of the intended service category (e.g. temperature sensing, motion sensing), data source (e.g., physical sensors, social media APIs), programming model (e.g. streaming process, batch process, SQL, NoSQL) and deployment node (e.g. Edge node, Cloud node). The CKB is intended to capture metadata and common information about classes having comparable application and resource requirements. Shared context knowledge is allowed to be customised and reused by IoT users. Contrarily, coordinating contexts with the CKRs can split these representations on the bases of satisfying services effectively.

Conclusions The components that form the conclusion of the generated recommendation rule is represented on the right-hand side of the ER diagram. CKR is suggested by the generated recommendation rules. The CKR can be easily deployed by the user using some specific configuration deployment engine, such as Docker. Sometimes, users may be required to submit knowledge representations to particular deployment engines and create some CKAs. For example, the recommender engine firstly generates an image using knowledge representation and then the user uploads the image to Docker API for deploying the particular Edge or Cloud services. The deployed resources configuration can be easily managed by users at any time.

3.4.2 Single Conclusion Ripple Down Rules

To model heterogeneous IoT resource configurations and to facilitate adequate reuse of existing CKRs, we employ a commonly used knowledge acquisition and maintenance

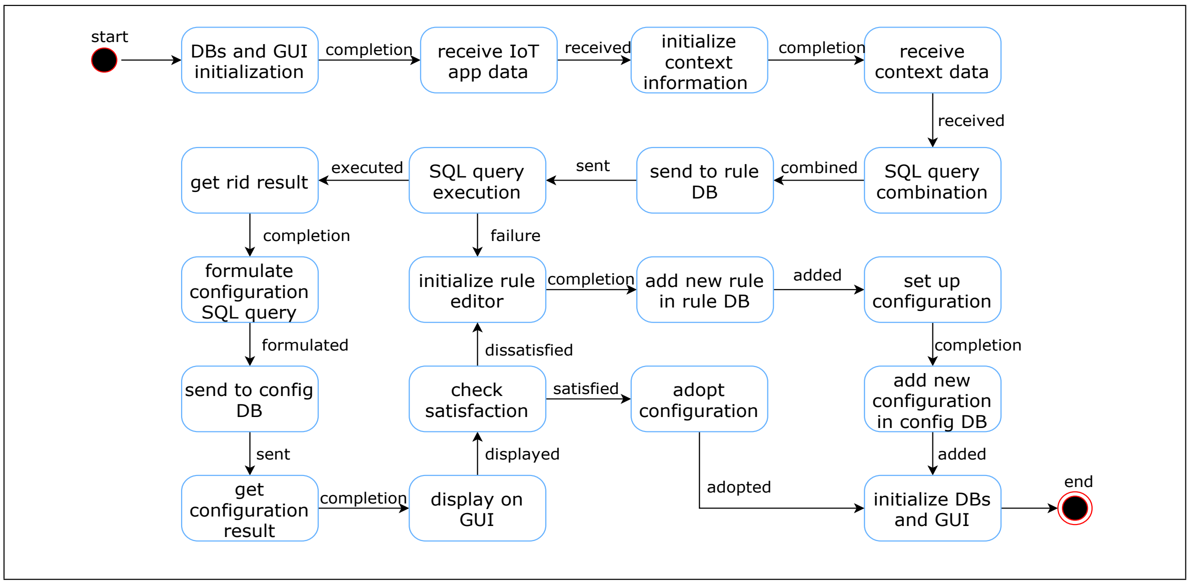


Figure 3.4: State transition diagram of IoT-CANE

approach, Ripple Down Rules [91]. The decision to choose RDR is because it enables the re-usability of the existing CKRs and CKAs. It also enriches the CKB by simply creating and attaching new rules to it. Many domains e.g. database cleansing, UI artefact reuse, NLP, etc. have successfully implemented the RDR technique. Based on our knowledge, RDR has not been adopted in the IoT resources configuration representation field.

Single conclusion RDR, multiple classification RDR and collaborative RDR are the common variations of RDR available [92]. The proposed IoT-CANE given in this chapter utilises a single conclusion ripple down rules technique that considers only one conclusion for given contextual information.

3.5 Design and Implementation

In this section, we present the implementation of system design, system workflow and the recommendation rule tree of proposed IoT-CANE.

3.5.1 System Design

Figure 3.4 depicts a state transition diagram of IoT-CANE which includes various states of IoT-CANE from start to end. Between every two states, a designed condition must be accomplished before moving the system state from one to another towards arrow direction.

In order to get the appropriate and accurate recommendations of resource configurations, we employ the knowledge from our conceptual model to specify each property in diverse IoT resources (physical resources, Edge resources and Cloud resources). Each property will be set in detail in the config editor module of IoT-CANE based on experts' experience to make sure these pieces of configurations are suitable. Also, these configurations may be optimised depending on the feedback from users. Only the administrator can operate the config DB and the rule DB in order to keep our databases stable and consistent. Such operations can be adding rule combinations/CKRs; deleting rule combinations/CKRs; modifying rule combinations/CKRs; changing association between rules and CKRs. It also protects the user from various operations.

Because each resource configuration combines a large set of attributes, IoT-CANE attempts to take over the massive number of choices associated with the attributes from the users to compensate for their lack of knowledge. With the aim of avoiding confusion for users, we design four context information categories (service category, data source, programming model and deployment node) to abstract the resource configurations in the IoT applications. The reason we choose these four context information categories is discussed below:

- *Service Category*: this is the main context information the user needs to input. For each IoT application, we abstract a list of services to indicate the currently available services in the specific IoT application. Along with the increasing number of services in the IoT application, the list will be updated based on both experts' experience and users' feedback. The demand of edge/cloud resources in diverse IoT applications are significantly different, that is why the choice of service category influences the CKR much, also that is the reason we choose it as the first context information category.

- *Data Source*: data source is the original "place" the raw data comes from. These places can be geographically distributed, and the raw data can be type non-sensitive. For instance, in a smart building application, the same temperature sensors deployed on different floors. These sensors can be considered as data sources which deployed geographical distributed. A temperature sensor which captures text data (temperature measurement), and a camera which captures image data can be considered as a different type of raw data. For a different type of data source and distributed data source, different CKRs will be recommended for each situation.

- *Programming Model*: programming model is another important context information. Widely used execution approaches include; stream process (e.g. Kafka Streams), batch process (e.g. Hadoop), SQL (e.g. MySQL), NoSQL (e.g. MongoDB). For each programming model, the recommended CKRs should be different for some specific properties. For example, the streaming process may require a higher rate of service availability and bigger bandwidth of network than batch process, because streaming processes data on a rolling window.

- *Deployment Node*: deployment node depicts the physical or virtual node for deployment. In IoT applications, such deployment nodes can be edge and cloud in general. To be more specific, edge node includes gateway, raspberry pi, mobile phone and etc.; cloud node includes private cloud, public cloud and various cloud platform offered by cloud providers (Amazon EC2, Microsoft Azure, etc.). The CKRs in edge node and cloud node are different. For example, the configuration in gateway may include DNS and IPv4 address setting, but these configurations are not available in Amazon EC2. Even the configurations between Amazon EC2 and Microsoft Azure are slightly different.

After capturing this context information, a relatively unique CKR can be recommended from IoT-CANE to cover users' requirement.

3.5.2 System Workflow

Next, we discuss the system workflow of IoT-CANE. Figure 3.5 shows the system workflow in a sequence diagram. First, the IoT-CANE graphic user interface (GUI) will get the IoT application set from the rule database (rule DB) after initialisation. When the GUI receives the set of IoT application and a specific IoT application is chosen,

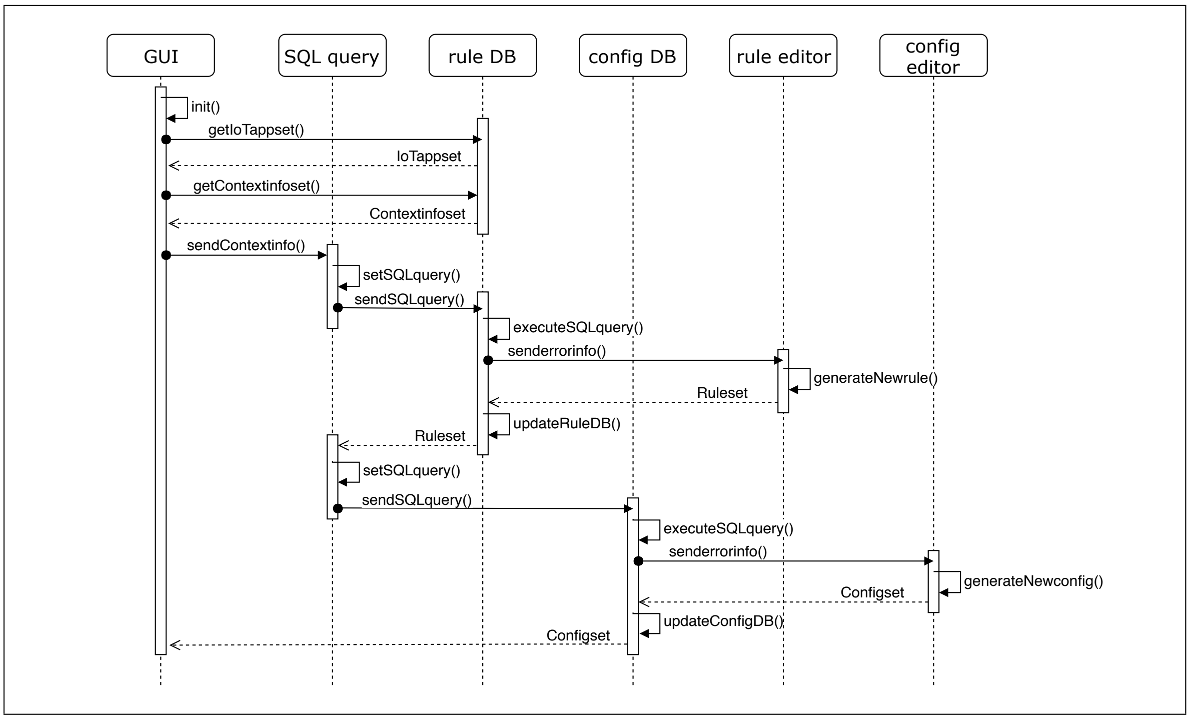


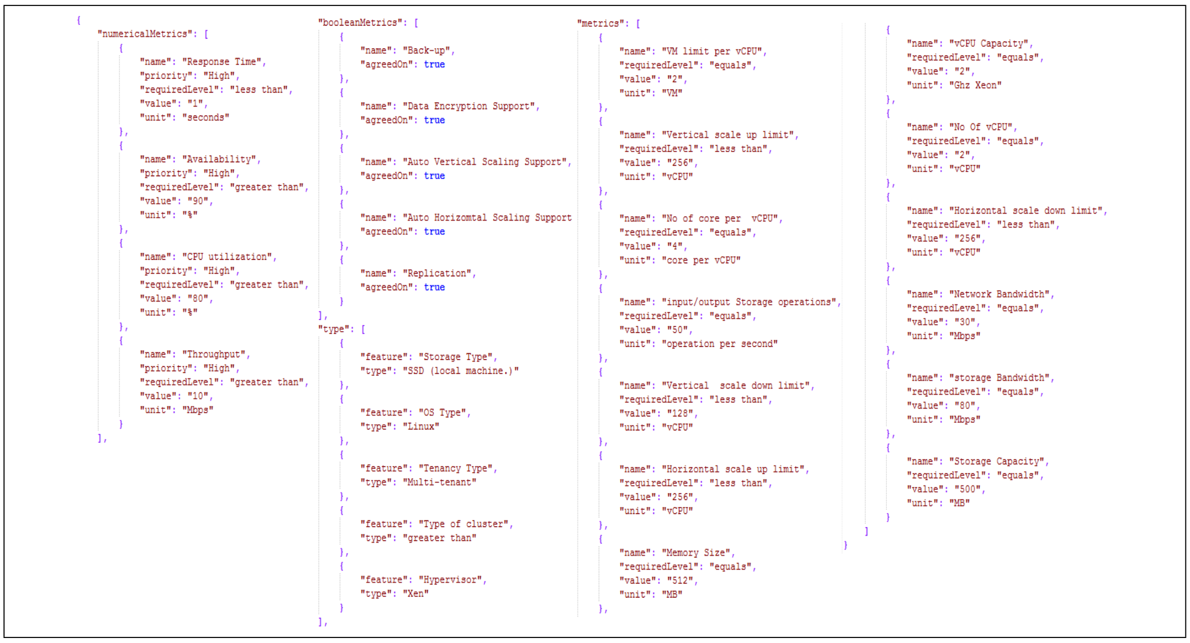
Figure 3.5: Sequence diagram of IoT-CANE

another get method will run in rule DB to get the context information set for updating context options in GUI. Afterward, the user will specify a set of context information including service category (sc), data source (ds), programming model (pm) and deployment node (dn). This piece of the message then is sent to the SQL query module to combine as a rule SQL query to run in the rule DB in order to index appropriate rules. In the worst case, nothing useful return, meanwhile, an error message with previously specified context information set is sent to the rule editor module. After a new rule set generated based on the given context information, it will be transferred to rule DB to update the index. Next, a configuration SQL query is composed based on the returned rule set in order to search relevant IoT resource configuration set. Another error message including the chosen rule will be sent to config editor module in the worst case as well. Then, a specific IoT resource configuration set associated with the sent rule will be generated based on the administrator's experience and expertise, also the given context information. After generation, the config DB module will receive the new resource configuration set and the update operation will run automatically. Then, this result will display on the graphic user interface for further deployment. Also, the user is attending to evaluate the return resource configuration set depending on their opinions.

The request for resource configuration representation recommendation in IoT-CANE is expressed as SQL queries. Next, we explain the basic steps which are executed for resolving a resource configuration representation recommendation request.

- System combines user's input context information to a temporary SQL query;
- The above temporary SQL query will be executed in the recommendation rule database to produce a possible result;
- Based on the result from the rule database, map the result to configuration representation database with rule number and show in the user interface;
- According to user satisfaction, new rule and configuration representation will be updated in the respective database after the administrator's operation.

A detailed example of the same scenario proposed in the first section will be explained below. When the householder purchases a new smart camera to deploy an intrusion detection service in his smart home application, he may feel confused on how to plug in the new smart camera to the smart home application and what are the configurations in each part of his smart home application. The problem here will be: which device's configuration will be changed and how to change them? The use can then use our IoT-CANE and input necessary contextual information under software guidance. In this case, he needs to choose 'IoT application' as 'Smart Home', 'Service Category' as 'Intrusion Detection', 'Data Source' as 'smart camera', 'Programming Model' as 'Streaming' and 'Deployment Node' as 'Gateway' respectively. The system then combines selected contextual information to a temporary SQL query along with SQL statement execution in the configuration knowledge base. Ideally, if there is an existing combination of selected contextual information in the CKB, the configuration knowledge representation of an intrusion detection service deployed with the smart camera on gateway will be displayed to the householder in the user interface. However, two unexpected situations will impact the result of the system process. One is the householder does not satisfied with provided CKR, which means the feedback from user is



```
{
    "numericalMetrics": [
        {
            "name": "Response Time",
            "priority": "High",
            "requiredLevel": "less than",
            "value": "1",
            "unit": "seconds"
        },
        {
            "name": "Availability",
            "priority": "High",
            "requiredLevel": "greater than",
            "value": "90",
            "unit": "%"
        },
        {
            "name": "CPU utilization",
            "priority": "High",
            "requiredLevel": "greater than",
            "value": "80",
            "unit": "%"
        },
        {
            "name": "Throughput",
            "priority": "High",
            "requiredLevel": "greater than",
            "value": "10",
            "unit": "Mbps"
        }
    ],
    "booleanMetrics": [
        {
            "name": "Back-up",
            "agreedOn": true
        },
        {
            "name": "Data Encryption Support",
            "agreedOn": true
        },
        {
            "name": "Auto Vertical Scaling Support",
            "agreedOn": true
        },
        {
            "name": "Auto Horizomtal Scaling Support
            "agreedOn": true
        },
        {
            "name": "Replication",
            "agreedOn": true
        }
    ],
    "type": [
        {
            "feature": "Storage Type",
            "type": "SSD (local machine.)"
        },
        {
            "feature": "OS Type",
            "type": "Linux"
        },
        {
            "feature": "Tenancy Type",
            "type": "Multi-tenant"
        },
        {
            "feature": "Type of cluster",
            "type": "greater than"
        },
        {
            "feature": "Hypervisor",
            "type": "Xen"
        }
    ],
    "metrics": [
        {
            "name": "VM limit per vCPU",
            "requiredLevel": "equals",
            "value": "2",
            "unit": "VM"
        },
        {
            "name": "Vertical scale up limit",
            "requiredLevel": "less than",
            "value": "256",
            "unit": "vCPU"
        },
        {
            "name": "No of core per  vCPU",
            "requiredLevel": "equals",
            "value": "4",
            "unit": "core per vCPU"
        },
        {
            "name": "input/output Storage operations",
            "requiredLevel": "equals",
            "value": "50",
            "unit": "operation per second"
        },
        {
            "name": "Vertical  scale down limit",
            "requiredLevel": "less than",
            "value": "128",
            "unit": "vCPU"
        },
        {
            "name": "Horizontal scale up limit",
            "requiredLevel": "less than",
            "value": "256",
            "unit": "vCPU"
        },
        {
            "name": "Memory Size",
            "requiredLevel": "equals",
            "value": "512",
            "unit": "MB"
        },
        {
            "name": "vCPU Capacity",
            "requiredLevel": "equals",
            "value": "2",
            "unit": "Ghz Xeon"
        },
        {
            "name": "No Of vCPU",
            "requiredLevel": "equals",
            "value": "2",
            "unit": "vCPU"
        },
        {
            "name": "Horizontal scale down limit",
            "requiredLevel": "less than",
            "value": "256",
            "unit": "vCPU"
        },
        {
            "name": "Network Bandwidth",
            "requiredLevel": "equals",
            "value": "30",
            "unit": "Mbps"
        },
        {
            "name": "storage Bandwidth",
            "requiredLevel": "equals",
            "value": "80",
            "unit": "Mbps"
        },
        {
            "name": "Storage Capacity",
            "requiredLevel": "equals",
            "value": "500",
            "unit": "MB"
        }
    ]
}
```

Figure 3.6: Example of Configuration Knowledge Representation

”not satisfied”. This situation will be met in some factors, e.g., users have knowledge with configuration knowledge representations, they know exactly what they need; users can hardly deploy their application based on the provided CKR, which means the CKR can not meet their requirement. Another situation is the temporary SQL query cannot acquire the corresponding result. In this case, our knowledge representation database cannot perform or acquire the particular CKR with given SQL query sentence because this given context combination has no history rule and corresponding CKR. With these two situations, the system will go to another layer to process the SQL request: the administrator can add a piece of the new rule in the configuration knowledge base with given contextual information in the rule editor module. Then he adds a corresponding CKR with his expertise and experience in the config editor module. After updating the CKB, the householder can review the returned resource configuration set formulated in JSON format on the IoT-CANE GUI. The example CKR in this scenario is shown in Figure 3.6 A set of deployment methods (e.g. Docker deployment) will be adopted using the recommended CKR in the further deployment. Up to now then, a new smart camera is discovered and added in the household’s proposed smart home application with an intrusion detection service.

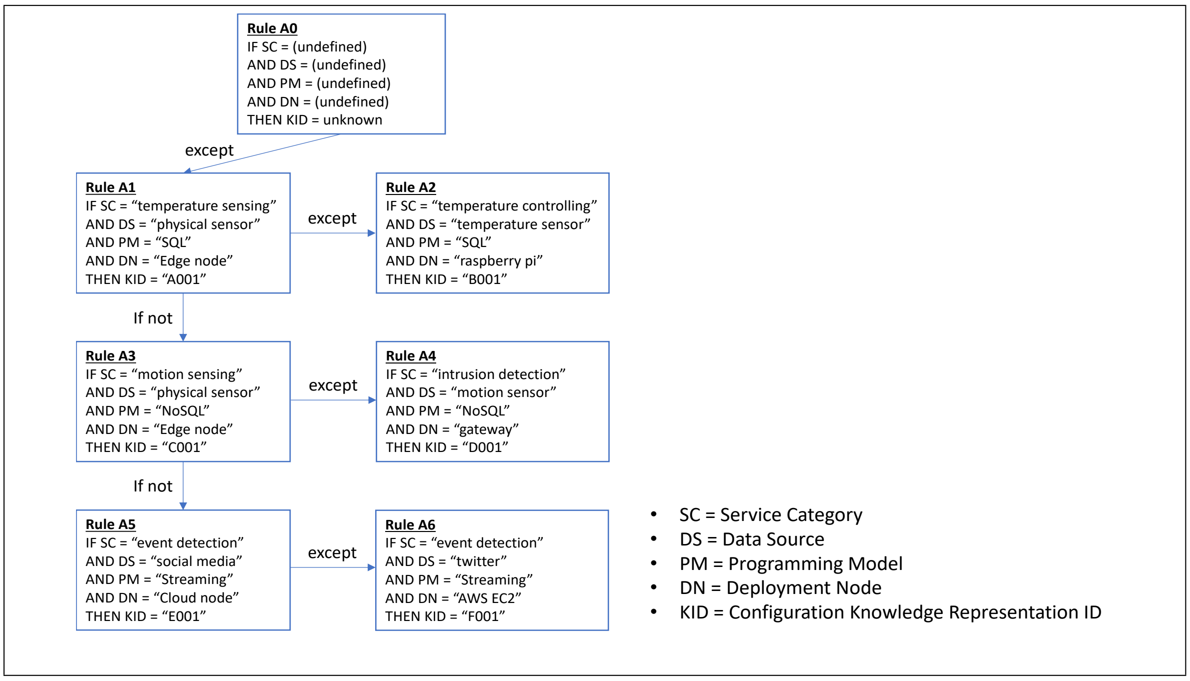


Figure 3.7: Example of recommendation rule tree structure

3.5.3 Recommendation Rule Tree

In the IoT-CANE, a tree architecture is adopted to organise numbers of recommendation rules and make connections among them. Figure 3.7 depicts the tree representation of the recommendation rules in the CKB. Default conclusion named ”unknown” is contained in Rule A0. This conclusion is suggested by the recommender system in the case when there is no specified input context; meaning that the service category, data source and other information is not provided in the input context. There are two possibilities “if not” (false) branches and “except” (true) branches for CKB to choose. When the IoT-CANE receives a new user context, the system passes the rule tree starting from root to the branches by checking the node is ”except” or ”if not” comparing with the recommendation rules. This step is repeated until there are no possible branches in the recommendation rule tree. The final conclusion received from the last ”except” node will be given to the user. A similar procedure is performed for each parameter (such as service category, data source, programming model and deployment node).

Let us consider an example of administrator for the CKB that wants to model an IoT resource configuration for a temperature controlling application as an Edge deployment. We assuming that the CKB does not have this service definition in any

Table 3.3: IoT recommender model parameters

Notations	Meaning
Query	A configuration selection query
SC = $\{sc_1,...,sc_n\}$	Set of n service categories
DS = $\{ds_1,...,ds_m\}$	Set of m data sources
PM = $\{pm_1,...,pm_o\}$	Set of o programming models
DN = $\{dn_1,...,dn_p\}$	Set of p deployment nodes
N	Number of rows in the database
M	Number of column in the database

available IoT applications. In this case, there is no Rule A2 for the recommendation rule tree structure in Figure 3.7. A query is generated for CKB to find a CKR that is related with service category “temperature sensing” and deployment node “Edge node” (Rule A1). But there is no such expert rule available generating from Rule A1. Hence, the administrator verifies the CKR linked with Rule A1 and confirms if this CKR is satisfactory for deploying a temperature controlling application. If the administrator updates the CKR, describing the required resource configuration for the temperature controlling application, the administrator adds one expert Rule A2 beneath Rule A1 and refers to the modified CKR as the result of Rule A2.

Summarising, the proposed IoT-CANE allows users to focus on the specific infrastructure requirements from the applications meanwhile the framework prevents users from any technical complexity of multiple IoT resource configuration solutions.

3.5.4 Computational Complexity

In this section, we discuss the computational complexity of the proposed system logic. The model parameters are discussed in detail in Table 3.3. For the worst case, our proposed system logic considers all the possible combinations (full CROSS JOIN). The total number of options varies based on the number of rows in each table. For the querying process, the upper bound complexity is given in equation (1):

$$O_{query}(sc_n \times ds_m \times pm_o \times dn_p) \tag{3.1}$$

However, the modern databases can use different techniques to reduce the computational complexity. For example, HASH JOIN and MERGE JOIN are widely used to reduce the computational complexity. They are $O(M+N)$ and $O(N*Log(N)+M*Log(M))$ respectively.

3.6 User Evaluation

In this section, we present the experimental setup and user evaluation on our proposed framework.

3.6.1 Experiment setup

We hosted the IoT-CANE on a local machine with 64bit Mac OS X operating system. The machine has the following hardware configuration: Processor (2.4 GHz Intel Core i5); Memory (8GB 1600 MHz DDR3); Graphics (Intel Iris 1536 MB); Storage (256 GB SSD). We chose MySQL database to implement the knowledge database.

3.6.2 User Evaluation

In order to evaluate the IoT-CANE, we performed a use case study to investigate the performance and acceptance of this system. Ten participants were invited to join the experiment. All participants are PhD students working in the Cloud Computing and Internet of Things area at Newcastle University. All of them had experience of deployment and configuration management in Cloud infrastructure and Edge devices. None of them had experience of using a recommender system for IoT resource configuration selection. The participants were asked to use the IoT-CANE under demonstration of an administrator.

They did a questionnaire after application testing. We gave nine questions to evaluate their experience and opinion of our IoT-CANE. The questions are listed below. Some of the results are shown in Figure 3.8.

- How satisfied are you with this system's ease of use?
- How often does our system freeze or crash?
- To what extent do you think that the recommendations are reasonable?

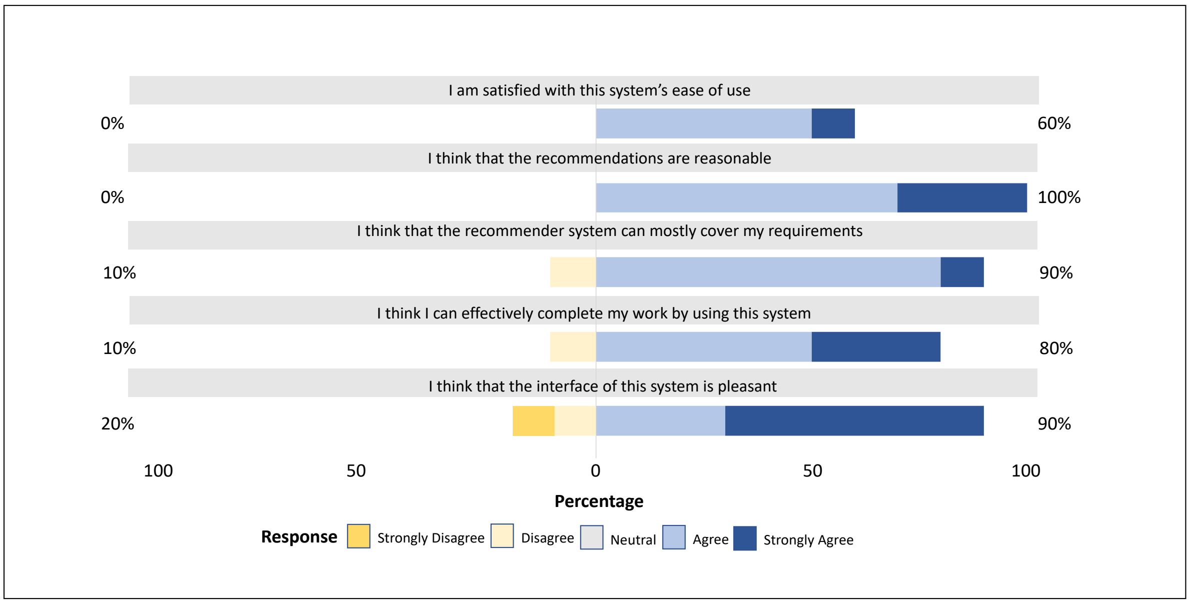


Figure 3.8: User survey result

- To what extent does the recommender system cover your requirements?
- To what extent do you think you can effectively complete your work using this system?
- Do you agree or disagree that the interface of this system is pleasant?
- How likely is it that you would recommend this software to a friend or group member?
- Overall, how satisfied or dissatisfied are you with our recommendations?
- How can we improve our recommender system?

As shown in Figure 3.8, most of the participants were satisfied with the IoT recommender system in the following ways: ease of use; reasonable recommendations; pleasant user interface etc. Based on their feedback, it can easily be concluded that the conceptual model covers the majority of resource configuration knowledge in IoT and the IoT-CANE can provide reasonable recommendations to IoT application users. However, not all of the participants think so. The final question in the survey asked the participants to give some suggestions to improve our IoT recommender system. Their suggestions can be categorised as follows:

- They wanted new features, such as automatic deployment;
- They suggested that the user interface could be more descriptive and user friendly;
- They suggested that the IoT recommender system could provide multiple choices of the configurations which can handle more scenarios.

Based on their suggestions, our IoT recommender system needs to be improved in two ways: (1) to provide a new service which can convert JSON format configuration files into Docker-readable configuration files, such as YAML format; (2) to provide an automatic deployment service based on container techniques.

3.7 Conclusion and Future Work

In this chapter, a model to construct the unified CKR for IoT infrastructure is presented. The framework comprises a CKR model for IoT infrastructure; a support recommender system for the recommendation of resource configuration in IoT; a service interface that converts context information to IoT resource configuration. The effectiveness of the proposed framework is evaluated using a proofed implementation and a user study. In the future, we are going to improve the IoT-CANE with some recommendation algorithms and also improve automatic deployment features in our recommender system.

4

IoTWC: Analytic Hierarchy Process Based Internet of Things Workflow Composition System

Contents

Summary

In this chapter, I present an *IoT workflow composition system (IoTWC)* to allow IoT users to pipeline their workflows with proposed IoT workflow activity abstract patterns. IoTWC leverages the analytic hierarchy process (AHP) to compose the multi-level IoT workflow that satisfies the requirements of any IoT application. Moreover, the users are befitted with recommended IoT workflow configurations using an AHP based multi-level composition framework. The proposed IoTWC is validated on a user case study to evaluate the coverage of IoT workflow activity abstract patterns and a real-world scenario for smart buildings. The comprehensive analysis shows the effectiveness of IoTWC in terms of IoT workflow abstraction and composition.

4.1 Introduction

The Internet of Things (IoT) has become an important component of leading software development and an integral part of our daily life. IoT is visible in many diverse communities such as smart buildings, smart agriculture, and smart industry [93]. With increasing demand and domain diversity, IoT applications become more complex in design, development, and deployment due to their multi-party Cloud/Edge/IoT resource makeup and the service level of agreement (SLA) requirements of their interconnections [94].

IoT applications can be modelled as a directed acyclic graph (DAG) with data transformation tasks as its nodes and data flow dependencies (or control flow dependencies) as its vertices. To be most useful in a modelling sense, we need an IoT workflow pattern that can be general enough to define any IoT application. Representing a generic IoT workflow pattern is intricate because of the heterogeneity of IoT infrastructure (IoT device, Edge device, and Cloud), coupled with application dependency on infrastructure and variability of data.

To narrow the accuracy of our modelling approach we provide a mechanism that converts the abstract workflow patterns into specific application workflow instances (see Figure4.1) (e.g., a Smart Home based on different QoS requirements such as IoT

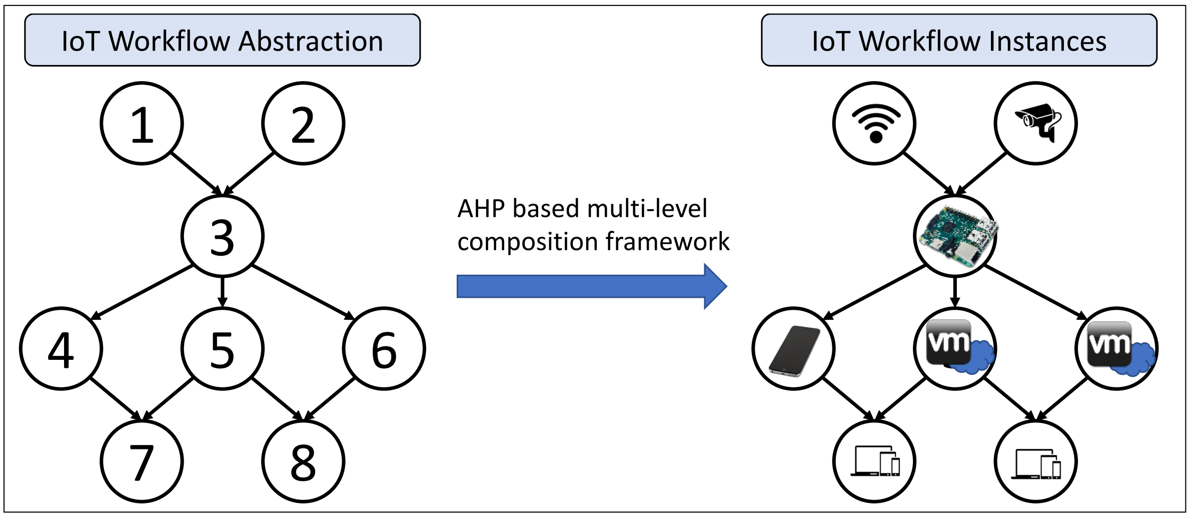


Figure 4.1: IoT Workflow Abstractions and Instances Mapping

device mobility, data privacy, and latency).

Identifying a set of reasonable IoT workflow activity patterns (provisioning an abstraction understandable to the engineer yet maintaining its usefulness for composition purposes), which covers IoT data transformation tasks and workflow activities, is the primary challenge in IoT workflow representation and composition. The aforementioned research challenges can be summarised to the following research questions:

- What are the abstract IoT patterns required to create generalised IoT applications across any domain?
- How to convert IoT workflow abstractions into IoT workflow instances specific for an IoT application based on the heterogeneous QoS requirements?

Traditional composition engines such as BPMN (Business Process Model and Notation) are specific only for business processes [95]. Workflow composition frameworks from academia [96–99] and industry (Calvin[30], Kubernetes [100], Amazon cloudFormation [101]) are also available but all of them are specific to a certain infrastructure or domain. Addressing IoT application users' subjective and objective opinions with abstract understanding of IoT workflow instances remains a challenge for a more holistic (cross-domain) approach. None of the existing frameworks are able to perform a generic IoT application composition.

To address the above challenges, we proposed and developed a novel composition framework, *IoTWC (IoT Workflow Composition System)*. Based on an extensive literature study and interviewing multiple people (domain experts/users), we provide basic activity patterns that are abstracted within IoTWC. IoTWC leverages the analytic hierarchy process (AHP) [102] to compose the multi-level IoT workflow that satisfy the requirements of any IoT application. The proposed IoTWC is validated on a user case study to evaluate the coverage of IoT workflow activity abstract patterns and a real-world scenario for smart buildings which show the effectiveness of IoTWC in terms of IoT workflow abstraction and composition.

The rest of this chapter is organised as follows. Section 5.2 discuss the related work. A detailed discussion about IoT workflow activity abstract patterns are presented in Section 4.3. Section 5.3 describes the IoT AHP model and multi-level composition framework in IoTWC while system design and implementation is discussed in Section 4.5. Section 5.5 presents the system evaluation with a user case study and a real world scenario validation followed by conclusion in Section 5.6.

4.2 Related Work

Application specification and composition is well-studied problem for general purpose applications. BPMN (Business Process Model and Notation) is a graph-oriented specification language representing business processes. As an implementation of BPMN for the web service domain, WS-BPEL (Web Services Business Process Execution Language) is an XML-based specification language that leverages web services to interact with each other in any web-based business approach. Glombitza [103] uses the BPEL for IoT application modelling to realise business processes. Domingos [104] defines IoT-aware business processes with the BPEL language to avoid increases in process complexity. XPDL (XMP Process Definition Language) is standardised by the Workflow Management Coalition (WfMC) to interchange the graphical business process workflow models to an XML-based model. Although these business workflow modelling approaches can present the flow of information, they do not model the specific data transformation tasks in IoT which are key to a successful IoT deployment and requires managing the correct configuration of devices and their usages.

Chen [105] proposed a RESTful web service to encapsulate heterogeneous IoT devices, in order to manage and compose IoT services for social network deployments. Urbieta [106] presented a context-aware web service description language based on adaptive service composition frameworks to support dynamic reasoning in IoT-based smart city applications. Chen [107] proposed trust management to support service composition applications in service-oriented architecture based IoT systems. Baker [108] developed a multi-cloud IoT service composition algorithm to create an energy-aware composition plan to fulfil user requirements. These works have addressed many IoT workflow and service composition research problems in a variety of application domains. However, there is no general approach to abstract the IoT workflow composition problems that can be useful across multiple IoT domains.

Many industry sectors also propose composition frameworks for supporting IoT application deployment. Da [28] applied TOSCA (Topology and Orchestration Specification for Cloud Application) in IoT to automate the deployment process of IoT applications based on the mosquito message broker. Persson [30] modelled IoT applications using four well-defined aspects: describe, connect, deploy and manage. This eased IoT application development and deployment processes for engineers. [100], [1] provided open-source approaches to benefit the design and deployment of IoT application workflows. However, these frameworks are designed for expert users having detailed background knowledge. This is not suitable for general users having little technical knowledge. Additionally, these frameworks do not focus on data transformation aspects of the IoT workflow composition process.

4.3 IoT Workflow Activity Abstract Pattern

Due to the heterogeneity of IoT applications, including their supporting Cloud/Edge data processing tasks, the traditional approach of presenting IoT workflow activities as a set of abstract patterns is difficult. Representation of IoT applications as a directed acyclic graph (DAG) simplifies this process by modelling each node as an IoT

[1]https://aws.amazon.com/cloudformation/

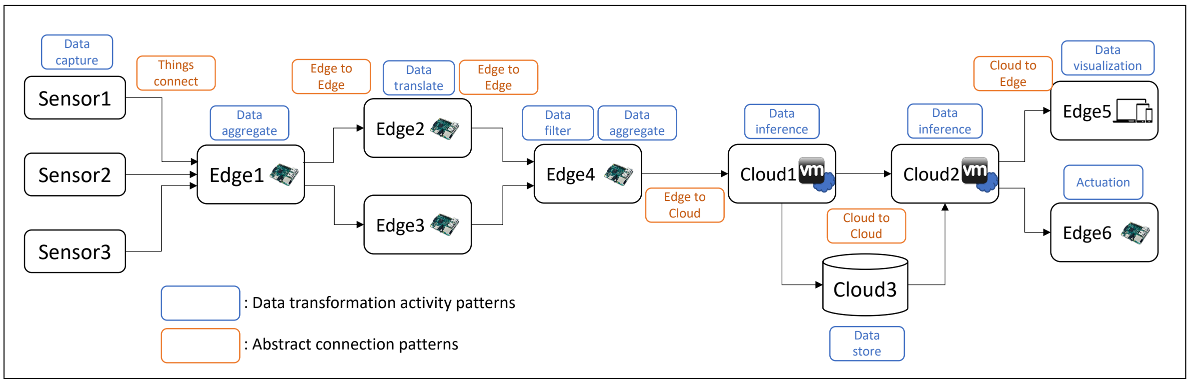


Figure 4.2: Example IoT workflow with abstract patterns

data transformation task. Thus, data is considered as a meta component in an IoT application. Considering a typical IoT application, the fundamental process consists of gathering and dissemination of data (e.g., capturing raw data from temperature sensors and storing this processed data into the database). In this typical example, we abstract IoT workflow activities as the following data transformation patterns: *Data Capture*, *Data Store*, *Data Inference*, *Data Filter*, *Data Aggregate*, *Data Visualisation*, *Data Translate*, and *Actuation*. Furthermore, we also consider the main abstract connection patterns in IoT: *Things Connect* and *Data Transfer* that consist of four sub-categories (Edge to Edge, Edge to Cloud, Cloud to Edge and Cloud to Cloud) in terms of sender and receiver of data. An example IoT workflow with proposed abstract patterns is shown in Figure 4.2.

After comprehensive literature reviews and interviews with IoT domain experts, we believe that our proposed IoT workflow activity abstract patterns cover the majority of the IoT data transformation tasks and associated data flows together with control flow requirements across multiple domains. The detailed evaluation will be discussed in section 5.5.

4.4 IoTWC: AHP-based Model and Multi-level Composition Framework

In this section, we describe the IoT AHP model and multi-level composition framework.

4.4.1 IoT Analytic Hierarchy Process Based Model

The complexity of IoT applications and their QoS requirements bring huge challenges for converting IoT workflow abstractions into IoT workflow instances. The fundamental aspect of this problem can be formalised as a Cloud/Edge resource Configuration Knowledge Representation (CKR) selection task. This selection task can be translated into a multi-criteria decision analysis problem with multiple alternative choices.

Analytic Hierarchy Process (AHP)[102] is an effective method for solving complicated multi-criteria decision making problems. In addition, AHP encourages decision makers to prioritise and identify how and when an appropriate decision is considered. The AHP can capture not only subjective, but also objective aspects of a decision by decreasing complicated decisions to a list of pairwise comparisons and then merge the results. Moreover, AHP adopts a technique to evaluate the consistency of the decision made by decision makers, resulting in the lowering of bias in the overall process under consideration.

In the AHP algorithm, a set of evaluation criteria and alternative choices need to be considered and identified by users. In addition, users also need to specify a weight between each two of the evaluation criteria based on their pairwise comparisons. Next, it will assign a score to each alternative choice based on users' pairwise comparisons. Finally, the algorithm will combine the criteria weights and the choice scores to generate a global ranking for alternatives. The ranking represents the sequence of each choice.

Figure4.3 illustrates the hierarchical representation of the CKR selection problem. Here, CKR selection is the primary goal and is based on three main criteria: Resource Cost, Resource QoS, Data. For each criterion, there is a set of sub-criteria prescribed (e.g., Hardware Cost, Hosting Cost and Network Cost are sub-criteria for Resource Cost; Reliability, Mobility, Heterogeneity, Scalability, Capability and Resource Availability are sub-criteria of Resource QoS). Based on our selection goal, we have a list of alternative choices among edge and cloud resources, such as CKR1 (high-performance edge resource, Raspberry Pi 4 model), CKR2 (low-performance edge resource, Raspberry Pi Zero model), CKR3 (high-performance cloud resource, AWS EC2 t2.xlarge)

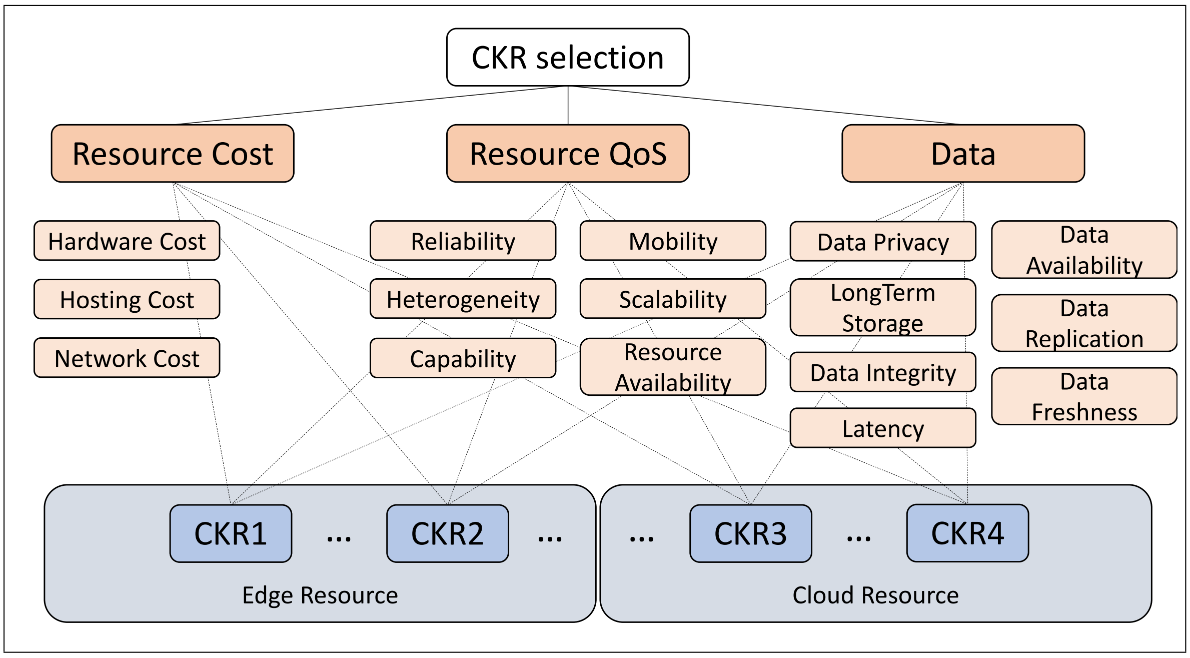


Figure 4.3: CKR Selection Hierarchy

and CKR4 (low-performance cloud resource, AWS EC2 t2.micro). When the IoT AHP model is enacted, IoTWC ranks all alternatives based on criteria weights between each two of the evaluation criteria. Users can retrieve the score and ranking, and the decision of CKR selection.

4.4.2 *Criteria Definition*

In this section, we discuss our criteria in more detail.

4.4.2.1 Resource Cost

Resource cost represents all financial commitments in the IoT application life cycle, like hardware, hosting, and network costs.

- *Hardware Cost*: Hardware cost includes all costs associated with hardware purchasing, such as IoT devices (e.g., sensor), computational devices (e.g., Raspberry Pi) and network devices (e.g., router) [109]. For each workflow activity, hardware costs depend on which resource is targeted. For example, users who plan to deploy their workflow activity in Edge resources may need to pay for Edge devices, such as a Raspberry pi; in contrast, workflow tasks located in Cloud resources can save this cost in hardware comparatively.

- *Hosting Cost*: Hosting cost represents the cost of provisioning the IoT application on hardware/software services enabling them available for use [109]. For Edge resources, hosting cost primarily include the maintenance cost, such as power consumption cost. Meanwhile, hosting cost in Cloud includes maintenance in various enabling services such as compute, storage, database, and network. Therefore, we consider a workflow activity deployed in Edge will be cheaper than Cloud in terms of hosting.

- *Network Cost*: Network Cost relates to the money spent on managing the networking and resource connections in Cloud and Edge applications [110]. For Cloud resources, the cloud providers offer networking services, like virtual networks, load balancing, application gateways, network monitoring infrastructures, and traffic managers. At the Edge, network cost mainly represents the bandwidth paid for. A major difference is that in Cloud based services such costs are paid on a pay-for-use basis where as in Edge scenarios pre-pay for services in advance is utilised.

4.4.2.2 Resource QoS

Resource QoS refers to the quality of service that both Edge and Cloud resources can provide. It consists of reliability, mobility, heterogeneity, scalability, capability and resource availability.

- *Reliability*: The probability that a service or a system can perform without any failures within a time interval is considered as reliable [111]. In workflow activity deployment scenarios, reliability also represents the high availability of Cloud and Edge resources. Cloud resources can provide full-stack solutions and comprehensive trouble-shooting and debugging services (increasing costs).

- *Mobility*: Mobility refers to the ability to migrate and transfer data, services and applications across Edge devices and Cloud servers [112]. Mobility also represents the ability for physical movement of Edge devices and Cloud datacenters.

- *Heterogeneity*: Heterogeneity represents the difference of hardware, software, infrastructures, architectures and technologies of both Cloud and Edge resources [113].

Numerous cloud providers offer services with different technologies and infrastructures. Many IoT device manufacturers together with their propriety solutions result in the Edge infrastructures that are highly heterogeneous.

- *Scalability*: Scalability is provided in two dimensions: Horizontal and Vertical [114]. Horizontal scalability indicates the ability to increase the same type of resources to satisfy load. For example, increasing the number of virtual machines and containers is a type of horizontal scalability. Vertical scalability indicates increasing the capability of an existing service, such as increasing CPU, memory and bandwidth of a virtual machine.
- *Capability*: Capability represents the ability to achieve a requirement [115]. Capability indicates the ability to integrate Edge or Cloud resources and the technologies to align with the users' strategic requirements.
- *Resource Availability*: The percentage of time that a user can access and operate a specific service is considered as resource availability [116]. Additionally, we can calculate the resource availability percentage by using total service time minus the time for which service is not available, then divided by total service time.

4.4.2.3 Data

Data indicates all data related criteria considered in our proposed model that affects the decision making, such as data privacy, data availability, long-term storage, data replication, data integrity, data freshness, and latency.

- *Data Privacy*: Privacy indicates the access control of data maintained by devices and services [117]. They need to remain in charge of their data in spite of third party data.
- *Data Availability*: Availability represents the ability to ensure data can be accessed when required [118]. In both Edge and Cloud resources, users expect to have complete access to their data at all times.
- *LongTerm Storage*: LongTerm storage describes the data that will be stored for a long period, usually in the data centre [119]. In the Cloud, the providers offer

various data storage services helping users to store data safely and continuously. For Edge devices, they may have their own storage for temporary data storage.

- *Data Replication*: Data replication provides the ability to store data in more than one database or network node [120]. This promotes data availability and reduces risks of data loss and, ultimately, failure.
- *Data Integrity*: Data integrity represents the accuracy, validity, and consistency of data over the whole life-cycle [118].
- *Data Freshness*: Data freshness indicates the recent nature of data in terms of generation and collection [121]. This helps reduce out-of-order date messaging. In IoT, freshness is a serious concern when dealing with in-stream analysis and management.
- *Latency*: Latency indicates the delay to message response time and the time for data transformation tasks [122]. Latency is influenced by the following factors: geographical location, bandwidth, computational power. For Cloud resources, higher bandwidth along with increased computational power can be employed to minimise latency. However, cloud data centers are always located in a specific geographic location which may be far from users' server (indicating there is little to be done regarding this aspect of latency). However, the geographic positioning of Edge devices can have a significant impact on latency (but they have limited ability to alter bandwidth or processing capabilities).

4.4.3 Multi-level Composition Framework

The composition framework of IoTWC works in twofold manner. It first applies AHP-based algorithm to combine both subjective and objective criteria functions for the selection of CKR and then take into account the financial budget to build an IoT application platform.

For the application of AHP-based algorithm, user need to provide the preference of various criteria (see Section 4.4.2). Consider there are $\mathcal{L}$ number of levels in the hierarchical representation of the problem and $\mathcal{R}$ is the set of criteria. The preference

Algorithm 1: Multi-level Composition

```
Input: $Pref_{kl}$ – preference of $k$th criteria over $l$th criteria, $\mathcal{R}$ – all criterion defined in AHP
       model, $r_{ij}$ – set of $i$ criteria in first level and $j$ sub-criteria, $Val_{r_{ij}o_n}$ – value for $r_{ij}$
       criteria and $o_n$ option, $\sigma_t$ – a ranking list for $t$th instance, $\theta_t$ – best ranking for $t$th
       instance, $\mathcal{B}$ – budget, $\mathcal{C}_u$ – cost of $u$th instance, $\mathcal{C}_{sum}$ – sum of instance cost
Output: $CKR$ – list of CKRs
1  construct $\mathcal{M}_{kl}$ using $Pref_{kl}$
2  if (!$Consistent(\mathcal{M})$) then
3      Notify user to enter new values
4      return -1
5  else
6      $\mathcal{W}$ = normalise (principle eigen vector ($\mathcal{M}$))
7  end
8  for each $r_{ij} \in \mathcal{R}$ do
9      multiple $\mathcal{W}$ and $Val(\mathcal{RO})$ to get $\sigma_{\mathcal{O}}$
10     select best ranking $\theta_t$ from $\sigma_{\mathcal{O}}$ for each instance
11     $CKR$ = sum of $\theta_t$ selection result
12 end
13 $\mathcal{C}_{sum} = \sum_1^u \mathcal{C}_u$
14 if $\mathcal{C}_{sum} > \mathcal{B}$ then
15     select second best ranking of $\sigma_t$ for each instance
16 else
17     return $CKR$
18 end
```

of criteria k with respect to l, $Pref_{kl}$ is provided following the Saaty scale as shown in table 4.2. Next, a comparison matrix $\mathcal{M}_s$ is constructed from the user's preferences following the rule given in equation 4.1.

$$\mathcal{M}_s = \begin{cases} 1, & \text{when } k = l \\ Pref_{kl}, & \text{when } k > l\,\} \\ 1/\mathcal{M}_{kl}, & \text{when } k < l \end{cases} \tag{4.1}$$

Since the user enter value may not be consistent, it is important to check the consistency of each comparison matrix $\mathcal{M}$ constructed using the user's preference values. If the comparison matrix is found inconsistent, user is notified to enter new values, otherwise, the weights are calculated. The weights $w_k \in \mathcal{W}$ for criteria k is calculated by normalizing the principle eigen vector of the respective matrix $\mathcal{M}$. Finally the rank vector $\sigma_{\mathcal{O}}$ of option $\mathcal{O}$ is calculated by multiplying the weight vector $\mathcal{W}$ with the normalized option values $Val(\mathcal{RO})$. This process is repeated for each instance t which can be later combined in the next step to select one complete application instance.

In the second level composition, we will calculate each IoT workflow activity cost based on the recommended configurations. For example, the budget column will show

Table 4.1: IoT AHP model parameters

Notations	**Description**
$\mathcal{L}$	Level of criteria in AHP
$\mathcal{R} = \{r_{11}...r_{ij}\}$	set of criteria
$Pref = \{Pref_{11}...Pref_{kl}\}$	set of preference of kth criteria over lth criteria
$\mathcal{M} = \{\mathcal{M}_1...\mathcal{M}_s\}$	set of s comparison matrix
$\mathcal{O} = \{o_1...o_n\}$	set of n options in AHP
$Val(\mathcal{RO}) = \{Val_{r_{11}o_1}...Val_{r_{ij}o_n}\}$	set of value for each criteria and options
$\mathcal{W} = \{w_1...w_q\}$	set of weight for q criteria
$\sigma_{\mathcal{O}}$	ranking list for option $\mathcal{O}$
θ_t	best ranking for tth instance
$\mathcal{B}$	budget for IoT application
$\mathcal{C} = \{c_1...c_u\}$	set of u instance cost
$\mathcal{C}_{sum}$	sum of instance cost
CKR	list of CKRs

Table 4.2: Relative Importance Value

Importance	**Value**
Equal important	1
Moderate important	3
Strong important	5
Demonstrated important	7
Extreme important	9
Intermediate	2,4,6,8

the cost when deploying such Edge/Cloud resources for a given period. IoTWC will compare the sum of the IoT workflow activity cost $\mathcal{C}_{sum}$ together with the user recommended budget $\mathcal{B}$ to determine which composed IoT workflow is acceptable. If the sum is over budget, the system will first display the cost result and recommend one or more CKRs. Therefore, IoTWC can perform multi-level compositions to make appropriate decisions under a certain budget and so better reflect a real-world scenario of deployment. The pseudo code of this multi-level composition algorithm is shown in Algo 1.

4.4.4 Computational Complexity Analysis

The AHP based IoT workflow composition problem appears complex due to the computational cost of the multi-level composition framework. The computational complexity of our approach is described in equation 4.2:

$$O(u \times n \times (i \times j^3 + i^3)) \tag{4.2}$$

In this equation, u represents number of instance in an IoT application, n is the number of options in each instance, $i \times j^3$ and i^3 indicates the complexity of calculating the weight vector for all criteria. However, in our case, i, j, u are concrete number based on given IoT application. Therefore, the final computational complexity can be simplified to $O(n \times i \times j^3)$. Based on the given O, we can expect IoTWC's complexity to be proportional to n, i and j.

4.5 System Design and Implementation

This section describes the architecture, workflow, and implementation of IoTWC.

4.5.1 System Architecture

Figure 4.4 shows the architecture of IoTWC. IoTWC is developed as a Web application. The system allows a user to pipeline their IoT application workflow with IoT workflow activities and abstract patterns, along with their specific IoT application requirements for each activity. More specifically, users can easily drag and drop the desired IoT workflow activity from the provided IoT abstraction patterns. The input criteria weights for each section are selected in the similar manner. IoTWC will compose a user's IoT workflow activities together with the recommended configurations. Information entered by users relating to patterns, weights and configurations are stored in an incremental knowledge base to be recalled when required.

The incremental knowledge base is maintained by a well-established system, IoT-CANE 3, which facilitates knowledge acquisition and maintenance. In this system, an incremental method is used to automate a configuration knowledge artefact suggestion based on user requirements within IoT resource configuration management. These recommended suggestions are generated based on users' context information and domain expert edits and modifications. Details of IoT-CANE can be found in Chapter 3. Finally, a composed IoT workflow coupled with configurations will be returned to users via the web interface.

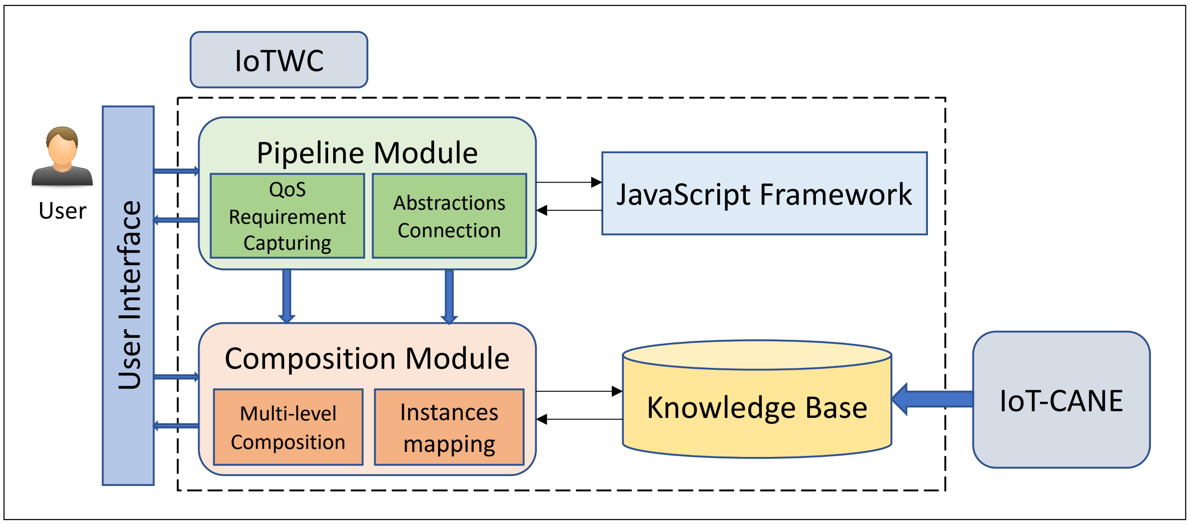


Figure 4.4: Schematic Design of IoTWC System

4.5.2 System Workflow

Fig. 4.5 shows the workflow of IoTWC. First, a user is required to input the necessary information through the web interface (step1), such as IoT workflow activities and their associated relationships. IoTWC then initialises by retrieving the appropriate abstract patterns associated with user input (step2). The user input IoT workflow activities are then sent to a javascript based pipeline module (step3) to allow the proposed IoT workflow pipeline to be displayed via the web interface (steps 4 and 5). The user will then specify the criteria weights for each proposed IoT workflow activity (step 6), e.g., weight between Resource Cost and Resource QoS. Once weightings are complete all information is sent to the composition module to allow IoT workflow configuration composition. In this module, an SQL query is constructed from IoT workflow activities (step 7) and their criteria weights suitable for the incremental knowledge base of IoT-CANE (step 8), allowing recommended configurations to be produced (steps 9 and 10). Finally, the recommended configurations are composed into a JSON format file which will be displayed in the web interface (steps 11 and 12).

4.5.3 System Implementation

The IoT workflow composition system is implemented and programmed in the Java programming language using the Spring framework 5.0 [2]The UI (user interface) is

[2]https://spring.io/

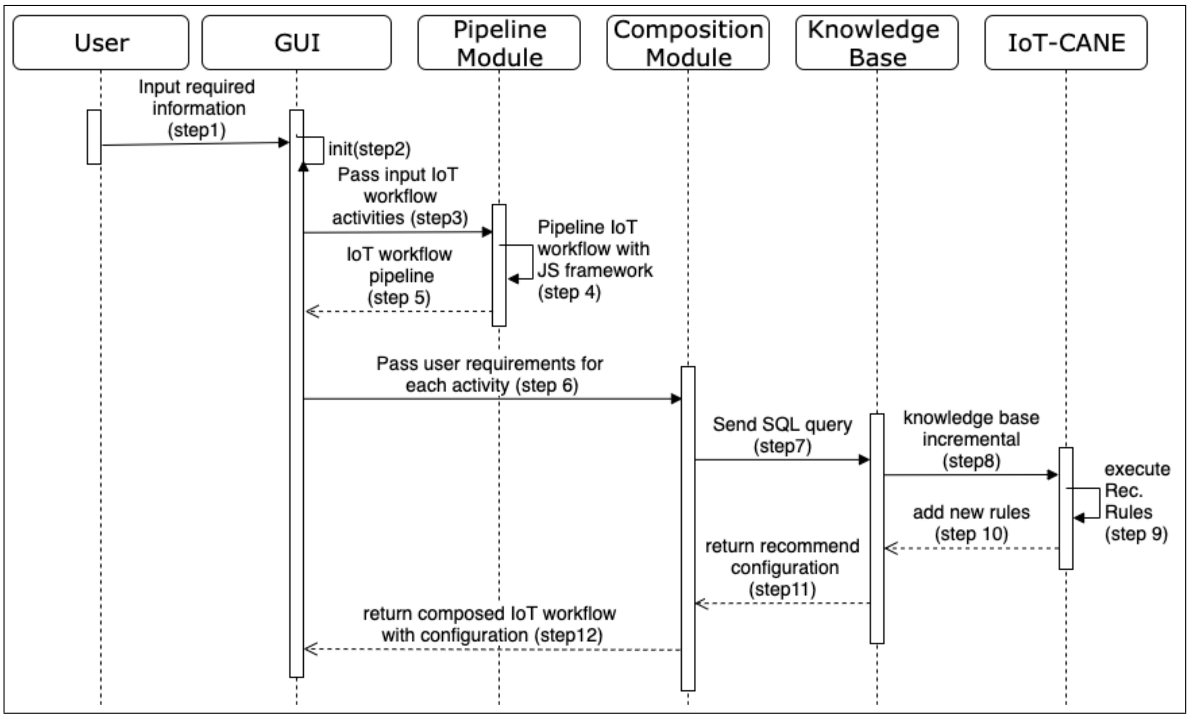

User
GUI
Pipeline Module
Composition Module
Knowledge Base
IoT-CANE
Input required information (step1)
init(step2)
Pass input IoT workflow activities (step3)
Pipeline IoT workflow with JS framework (step 4)
IoT workflow pipeline (step 5)
Pass user requirements for each activity (step 6)
Send SQL query (step7)
knowledge base incremental (step8)
execute Rec. Rules (step 9)
add new rules (step 10)
return recommend configuration (step11)
return composed IoT workflow with configuration (step12)

Figure 4.5: IoTWC System Sequence Diagram

created in HTML5, CSS, and JavaScript to ensure ”easy to use” website design principles. Our system is designed to help users to pipeline their IoT application workflow into the desired configuration. In addition, this also benefits mapping users’ abstract IoT application requirements to real IoT activity instances by applying the AHP based multi-level composition framework. Two main modules (JavaScript (JS) based pipeline module and AHP based composition module) used in our system are now described.

JS based pipeline module. IoTWC allow users to drag and drop IoT workflow activity abstract patterns to position them as required with the aid of the JS-based pipeline module. This module is implemented with GoJS [3],a JS library for building interactive diagrams and graphs. We present a set of IoT workflow activity abstract patterns as reusable components in the system. These components can be freely dragged from the left side of the interface and dropped on the blank workspace on the right side. Links can be built between each IoT workflow activity via mouse clicks, representing connections between Cloud and Edge resources.

AHP based composition module. When the planned IoT workflow activity is clicked by a user, a list of criteria indicating available pairwise comparisons are shown.

[3]https://gojs.net/latest/index.html

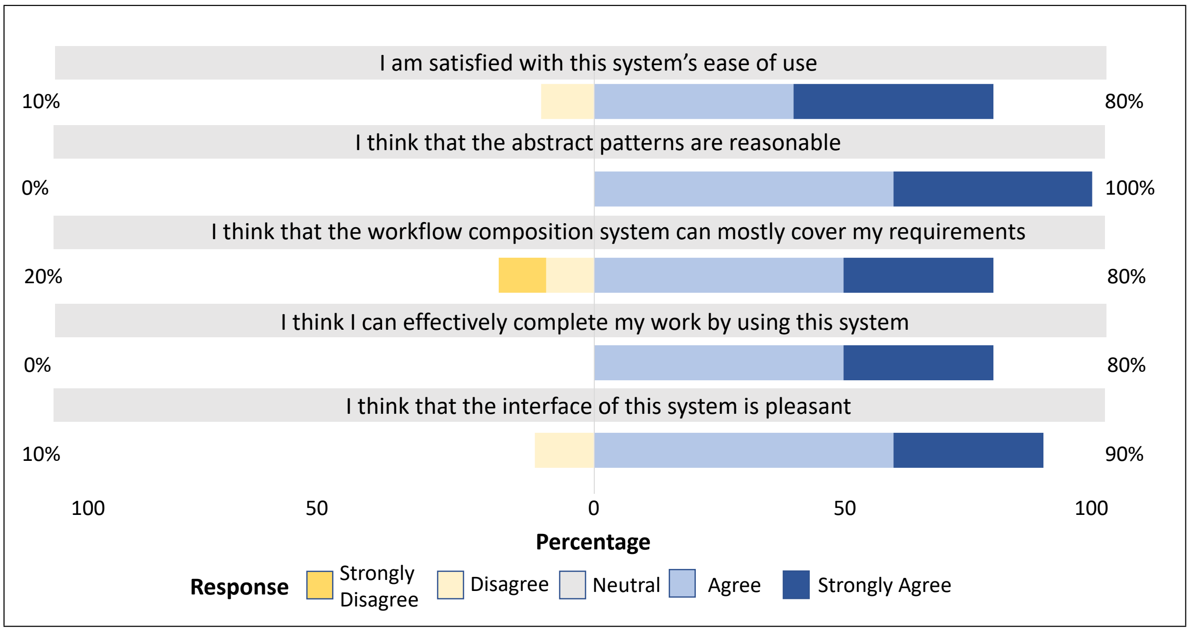


Figure 4.6: User Case Study Result

In the composition module, these pairwise comparisons capture the subjective and objective opinions from users to execute decisions by applying the IoT AHP algorithm. The composition module will calculate the ranking for the alternatives based on user supplied criteria weightings, then compose the appropriate SQL query to retrieve recommended configurations from the knowledge base (IoT-CANE). A Java library is used for vector calculations and score generation. Our implementation provides a simple and easy way for novice users to use IoTWC to display the composed results.

4.6 System Evaluation

This section describes the user case study and real world validation of our proposed IoTWC system.

4.6.1 User Case Study

This sub-section describes the experimental setup and user evaluation from our user case study.

4.6.1.1 Experiment Setup

As IoTWC is implemented in Java it can be composed into a JAR file to execute over a variety of environments, such as Windows, Linux, and MacOS. In this scenario, we host our workflow composition system on a MacBook Pro with MacOS operating system. The machine has the following hardware configuration: 1.4GHz Quad-Core Intel Core I5 processor, 16GB memory, Intel Iris Plus 1536MB Graphics and 512GB SSD storage. We run our tool using Visual Studio build environment Code[4] with Java and Spring Boot extensions. We use a MySQL[5] database as our data management tool. IoTWC is an open-source system and the current version of code is available on github[6].

4.6.1.2 User Evaluation

In order to evaluate IoTWC, we perform a use case study to ascertain acceptance and performance. We invited twelve participants, who are current Ph.D. or Masters students studying Cloud Computing and/or Internet of Things at Newcastle University. All of these participants have experience and knowledge in Cloud and Edge resource management and deployment. They do not have experience in IoT workflow composition.

After using IoTWC, the participants were asked to complete a questionnaire. Nine questions are used to investigate users' opinions regarding their experiences of IoTWC, listed below.

- How satisfied are you with this system's ease of use?
- How often does the system freeze or crash?
- To what extent do you think that the abstract patterns are reasonable?
- To what extent do the abstract patterns cover your requirements to pipeline IoT workflow?

[4]https://code.visualstudio.com/
[5]https://www.mysql.com/
[6]https://github.com/frankleesd/iot_workflow_composer

- To what extent do you think you can effectively complete your composition work using this system?
- Do you agree or disagree that the interface of this system is pleasant?
- How likely is it that you would recommend this software to a friend or group member?
- Overall, how satisfied or dissatisfied are you with the workflow composition system?
- How can we improve our IoT workflow composition system?

We chose five questions to show in Figure 4.6. As shown in this figure, most of the users were satisfied with IoTWC in areas such as reasonable abstract patterns, and pleasant user interface. According to the feedback, we can summarise that the IoT workflow activity abstract patterns can cover the majority of IoT workflow composition requirements. However, not all of the participants were fully satisfied. Based on the feedback, we can improve IoTWC in the coverage of some alternatives and other criteria.

4.6.2 Scenario Validation

This sub-section will validate the effectiveness of the proposed IoTWC using a real world smart building IoT application scenario highlighting the typical usage in an industry style project setting.

4.6.2.1 Scenario Description

In a real world smart build scenario, a user plans to deploy a smart IoT application that can capture relevant data from different room sensors to provide the status of the building. Basic sensors can provide temperature level, humidity level, and CO2 level while more advanced sensors can provide images and other useful information. This smart building IoT application workflow is represented in Figure 4.7. In this application, a user wishes to capture CCTV and a variety of other sensor data from rooms

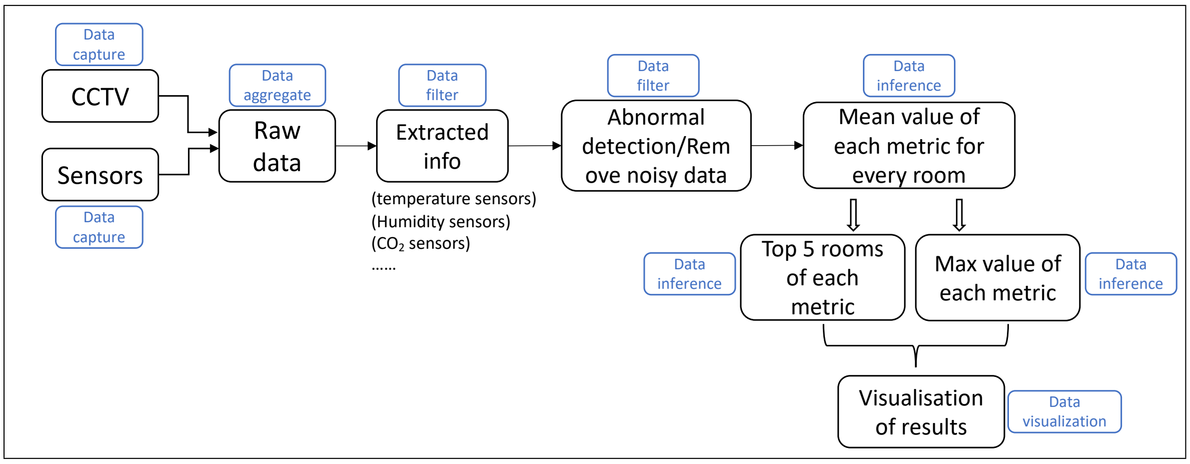
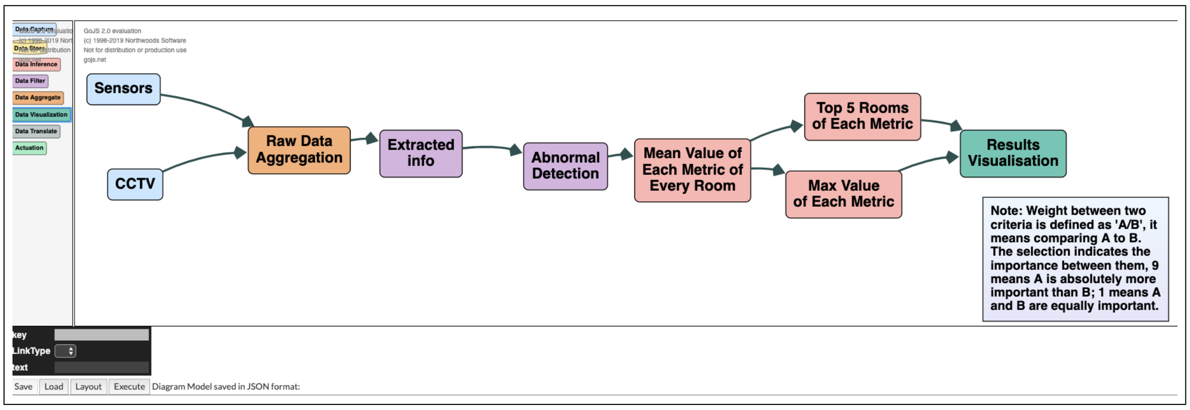


Figure 4.7: Smart building scenario workflow



Note: Weight between two criteria is defined as 'A/B', it means comparing A to B. The selection indicates the importance between them, 9 means A is absolutely more important than B; 1 means A and B are equally important.

Diagram Model saved in JSON format:

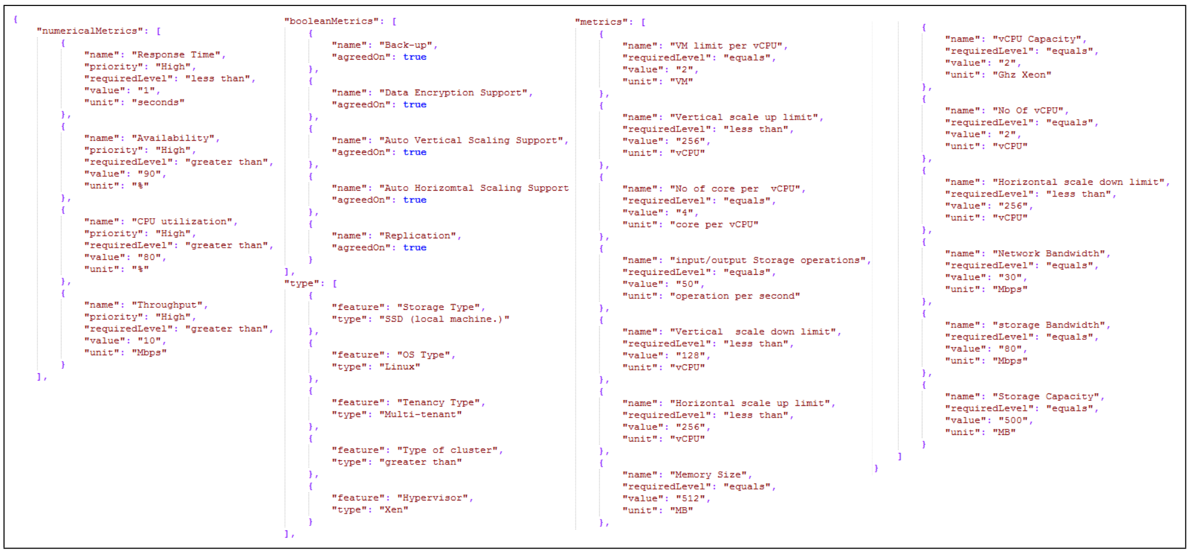
Figure 4.8: IoTWC running interface and pipelined workflow

while including novel data management/filtering possibilities utilising Edge devices, like a Raspberry Pi. This allows the extraction of raw data into different useful data sets, such as temperature data set, humidity data set and CO2 data set. Abnormal detection and noisy data removal is performed before data inference. In this case, the user plans to calculate the top 5 rooms of each metric and the associated max value of each metric, together with provisioning visualisation of the results.

4.6.2.2 Scenario Validation

First, the user can pipeline the smart building application workflow in IoTWC with given abstract patterns. The pipelined workflow is shown in Figure 4.8

When this smart building workflow pipeline is assessed by IoTWC, the user specifies a set of criteria weights in each IoT workflow activity which allows to capture a user's subjective and objective opinion according to his/her requirements. In addition, the



```
{
    "numericalMetrics": [
        {
            "name": "Response Time",
            "priority": "High",
            "requiredLevel": "less than",
            "value": "1",
            "unit": "seconds"
        },
        {
            "name": "Availability",
            "priority": "High",
            "requiredLevel": "greater than",
            "value": "90",
            "unit": "%"
        },
        {
            "name": "CPU utilization",
            "priority": "High",
            "requiredLevel": "greater than",
            "value": "80",
            "unit": "%"
        },
        {
            "name": "Throughput",
            "priority": "High",
            "requiredLevel": "greater than",
            "value": "10",
            "unit": "Mbps"
        }
    ],
    "booleanMetrics": [
        {
            "name": "Back-up",
            "agreedOn": true
        },
        {
            "name": "Data Encryption Support",
            "agreedOn": true
        },
        {
            "name": "Auto Vertical Scaling Support",
            "agreedOn": true
        },
        {
            "name": "Auto Horizomtal Scaling Support
            "agreedOn": true
        },
        {
            "name": "Replication",
            "agreedOn": true
        }
    ],
    "type": [
        {
            "feature": "Storage Type",
            "type": "SSD (local machine.)"
        },
        {
            "feature": "OS Type",
            "type": "Linux"
        },
        {
            "feature": "Tenancy Type",
            "type": "Multi-tenant"
        },
        {
            "feature": "Type of cluster",
            "type": "greater than"
        },
        {
            "feature": "Hypervisor",
            "type": "Xen"
        }
    ],
    "metrics": [
        {
            "name": "VM limit per vCPU",
            "requiredLevel": "equals",
            "value": "2",
            "unit": "VM"
        },
        {
            "name": "Vertical scale up limit",
            "requiredLevel": "less than",
            "value": "256",
            "unit": "vCPU"
        },
        {
            "name": "No of core per  vCPU",
            "requiredLevel": "equals",
            "value": "4",
            "unit": "core per vCPU"
        },
        {
            "name": "input/output Storage operations",
            "requiredLevel": "equals",
            "value": "50",
            "unit": "operation per second"
        },
        {
            "name": "Vertical  scale down limit",
            "requiredLevel": "less than",
            "value": "128",
            "unit": "vCPU"
        },
        {
            "name": "Horizontal scale up limit",
            "requiredLevel": "less than",
            "value": "256",
            "unit": "vCPU"
        },
        {
            "name": "Memory Size",
            "requiredLevel": "equals",
            "value": "512",
            "unit": "MB"
        },
        {
            "name": "vCPU Capacity",
            "requiredLevel": "equals",
            "value": "2",
            "unit": "Ghz Xeon"
        },
        {
            "name": "No Of vCPU",
            "requiredLevel": "equals",
            "value": "2",
            "unit": "vCPU"
        },
        {
            "name": "Horizontal scale down limit",
            "requiredLevel": "less than",
            "value": "256",
            "unit": "vCPU"
        },
        {
            "name": "Network Bandwidth",
            "requiredLevel": "equals",
            "value": "30",
            "unit": "Mbps"
        },
        {
            "name": "storage Bandwidth",
            "requiredLevel": "equals",
            "value": "80",
            "unit": "Mbps"
        },
        {
            "name": "Storage Capacity",
            "requiredLevel": "equals",
            "value": "500",
            "unit": "MB"
        }
    ]
}
```

Figure 4.9: Configuration Knowledge Representation Example

user needs to specify a budget for the smart building application. For example, in Raw Data Aggregation workflow activity, a user needs to specify a comparison weights between Resource Cost, Resource QoS, and Data. When these weight information is typed in, a comparison matrix is generated as follow:

$$CKR_{Selec} = \begin{array}{c|ccc|} & R_{Cost} & R_{QoS} & Data \\ R_{Cost} & 1 & 7 & 9 \\ R_{QoS} & 1/7 & 1 & 3 \\ Data & 1/9 & 1/3 & 1 \end{array}$$

The computation of this comparison matrix can give a ranking for Resource Cost, Resource QoS, and Data.

Additionally, each criteria has some sub-criteria for decision making. For example, a comparison matrix for Hardware, Hosting, and Network Costs is generated as follow:

$$R_{Cost} = \begin{array}{c|ccc|} & C_{Hardware} & C_{Hosting} & C_{Network} \\ C_{Hardware} & 1 & 1/3 & 9 \\ C_{Hosting} & 3 & 1 & 5 \\ C_{Network} & 1/9 & 1/5 & 1 \end{array}$$

These two comparison matrix are generated based on the information provided by the users. In the meantime, the comparison matrix between criteria and alternatives are defined and composed by domain experts. For example, considering scalability sub-criteria under resource QoS, Cloud resource may get more weight than Edge resource due to the scalability and services offered by cloud providers. The comparison matrix is shown below:

$$Scalability = \begin{array}{c|cccc} & Cloud_1 & Cloud_2 & Edge_1 & Edge_2 \\ \hline Cloud_1 & 1 & 1 & 9 & 9 \\ Cloud_2 & 1 & 1 & 9 & 9 \\ Edge_1 & 1/9 & 1/9 & 1 & 1 \\ Edge_2 & 1/9 & 1/9 & 1 & 1 \end{array}$$

Other comparison matrix are constructed in the similar manner. As a result of AHP execution, a list of best ranking for each instance are calculated by previous processes. Once the 'execute' button is clicked the IoT AHP algorithm starts querying the knowledge base to gain a set of appropriate configuration knowledge representations for each IoT workflow activity. The results are displayed to the user for further consideration. An example JSON format CKR result of one IoT workflow activity is shown in Figure 4.9.

4.7 Conclusion and Future Work

IoT workflow composition is a complex problem due to the heterogeneity of Cloud and Edge resources and data type diversity. We proposed and developed a novel AHP based multi-level composition framework and IoTWC system. With IoTWC, IoT application users can easily pipeline and compose IoT workflow application with recommended activity configuration knowledge representations under a certain budget. The results are investigated and validated in a real world smart building scenario. The results can be further utilised by user to deploy the workflow instance while making a balance between the performance and budget for each workflow activity. In the future, we plan to design and develop a deployment module to integrate with IoTWC, in order to automate IoT application development life cycle. We will provide an extension of IoTWC to cover deployment and orchestration processes in IoT applications.

5

A Fault-Tolerant Workflow Composition and Deployment Automation IoT framework in a Multi-Cloud Edge Environment

Contents

Summary

With the increasing popularity of IoT application, e.g., smart home, smart manufacturing, the importance of underlying IoT system availability, safety, reliability and maintainability become crucial in IoT application development processes. IoT applications are expected to continuously provide reliable services and features which put the fault-tolerance mechanism as priority. Current IoT fault-tolerant systems are designed to overcome any faults caused by human activities and physical errors in order to reserve the correct IoT workflow execution. However, addressing fault-tolerance interaction in multi-cloud edge environment and failed service deployment automation remain challenges. This chapter proposes a novel fault-tolerant model that offers the self-detection and automatic recovery of faults to increase IoT applications' reliability to address the infrastructure level failure in the heterogeneous IoT environments. Based on the proposed model, a fault-tolerant workflow composition and deployment automation system is developed and implemented. This system utilises a layered architecture and a time-dependent failure model to offer deployment automation and infrastructure recovering. The efficiency and effectiveness of the proposed system are validated and evaluated with a real-world IoT application. The evaluation result shows that the proposed model can reduce system fail rate with recovery algorithm enabled. Moreover, our recovery strategies reduce the cost of on-demand backup nodes significantly.

5.1 Introduction

The Internet of Things (IoT) is mainly driven by data which is transferred between resources including cloud, edge and IoT devices. It raises the importance of IoT systems in terms of availability, safety, reliability and maintainability. Reliability is a primary aim to implement for IoT applications concerned with Quality of Service (QoS). Reliability is threatened by the occurrence of failures where an IoT system can hardly offer potential services. There are three main methods to mitigate failures including fault correction, fault avoidance and fault tolerance, where fault tolerance refers to detect and recover faults in runtime. In an IoT environment, applications

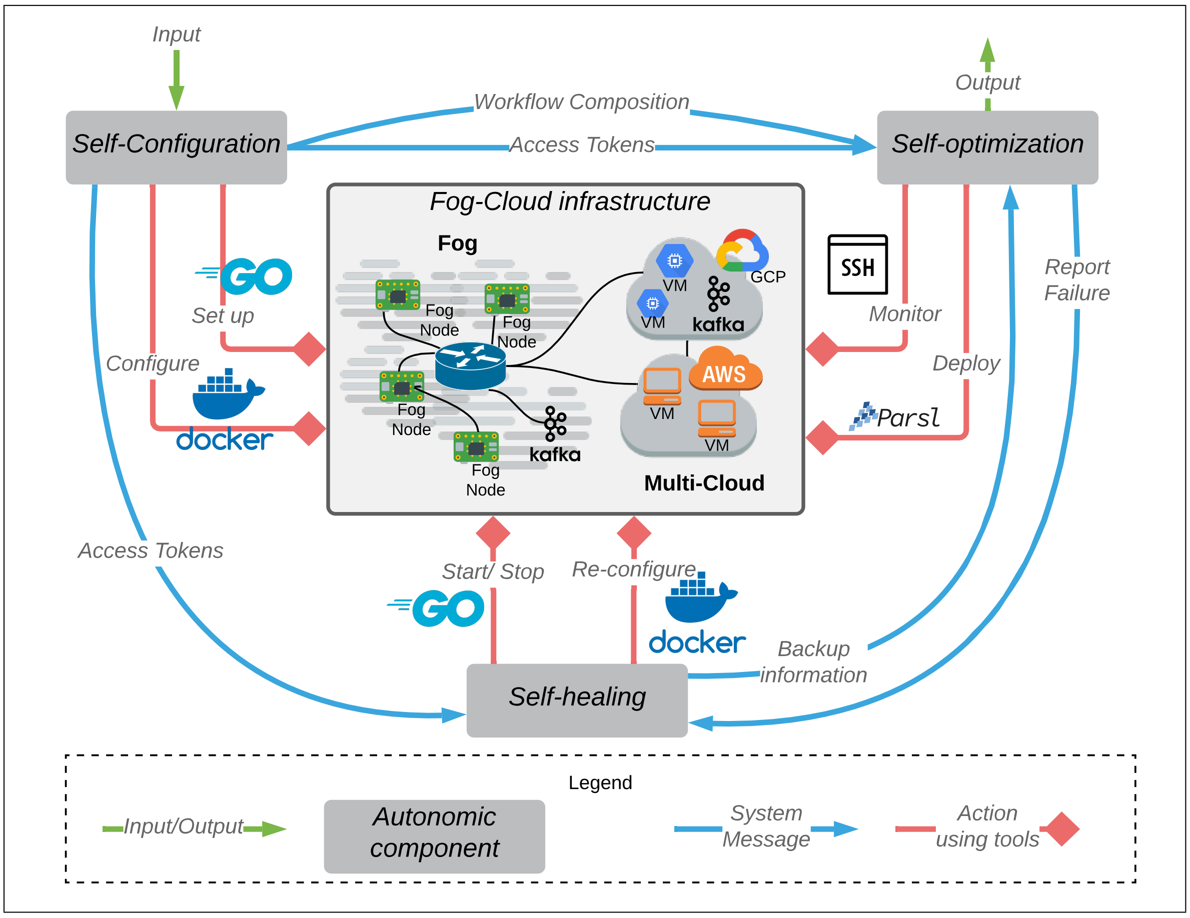


Figure 5.1: System Overview of Proposed Approach

are expected to continuously provide reliable services and features which put fault-tolerance mechanism as priority.

Consider that the Newcastle city council bids for a new IoT project for flood forecasting. Their engineers plan to develop and deploy a flood forecasting application which provides real-time rainfall map and main road risk level to avoid flood damage in the city. In this application, the raw streaming rainfall data captured by CCTV and sensors around the city is supposed to be delivered and analysed by a given rainfall model and flood forecasting model. The infrastructure for deployment includes CCTV, rainfall sensors, Raspberry Pi as edge devices and virtual machines on the cloud purchased and subscribed. Due to the QoS (end-to-end latency and reliability) and the city council budget limits, infrastructure selection becomes a challenge in this project. Meanwhile, to avoid infrastructure failure, a robust multi-level reconfiguration system is necessary. Moreover, a centralised deployment agent who can deploy and manage every task across different IoT application layers will ease the work.

This chapter plans a framework that enables a user to compose any IoT application with defined QoS parameters, then automatically set up the infrastructure according to the recommended configuration, as shown in Figure 5.1. The system is divided into three components: Self-configuration, Self-optimisation and Self-healing. Self-configuration takes user QoS requirements and a workflow model of an IoT application, and then it returns ready to use infrastructure. Moreover, the Self-configuration automatically sets up the infrastructure and sends the access tokens to Self-optimisation and Self-healing. Then, Self-optimisation loads configuration operates and monitors infrastructure according to QoS. During failure events, Self-optimisation reports Self-healing with system failures. Self-healing component restores operation to normal by backing up the infrastructure with resources and recover the faulty ones.

During the evaluation process, an IoT application is utilised to test the functionality of the proposed framework that solves challenging issues associated with developing and operating IoT applications, such as avoiding dependency issues using container technology. The experiment contains a platform and operates on raw streaming data and analyses on a given ML model near the edge. Moreover, multi-cloud infrastructure supports computing power, e.g., additional storage and processor type (FPGA or GPU). The application has chained service functions in the form of a Directed Acyclic Graph (DAG). The functions apply logical data operation, machine learning (ML) model training or ML prediction. Each service function has its execution requirements, which will determine the place of execution, i.e., cloud or edge.

The rest of this chapter is organised below: Section 5.3 explains the detail of the IoT fault-tolerance model. Section 5.4 illustrates the layered system architecture and Section 5.5 presents the experiment setup and results analysis. Before the conclusion, a set of related and relevant work is discussed in Section 5.2.

5.2 Related Work

In this section, we discuss the related work on workflow composition, deployment automation and fault-tolerance of IoT applications.

Workflow composition is mainly well-studied in general web services and applications.

BPEL (Business Process Execution Language) and WSDL (Web Service Definition Language) are popular XML-based workflow composition languages that are widely used in enabling web services interaction and communication.

Academic researchers have proposed many solutions and architectures for IoT workflow composition and deployment problems. A RESTful based web service is designed in [105] to encapsulate heterogeneous IoT devices. In order to support dynamic reasoning in smart city applications, a context-aware web service description language is presented in [106] based on adaptive service composition frameworks. A trust management framework is proposed in [107] to provide service composition in service-oriented IoT architectures. [108] developed a multi-cloud IoT service composition algorithm to create an energy-aware composition plan to fulfil user requirements. These works have addressed many IoT workflow and service composition research problems in a variety of application domains. However, there is no general approach to abstract the IoT workflow composition problems that can be useful across multiple IoT domains. Meanwhile, composition frameworks which support IoT application deployment are also well-defined in industry. OpenTOSCA applied TOSCA (Topology and Orchestration Specification for Cloud Application) in IoT to automate the deployment process of IoT applications based on the mosquito message broker. Calvin modelled IoT applications using four well-defined aspects: describe, connect, deploy and manage. This eased IoT application development and deployment processes for engineers. Kubernetes provided open-source approaches to benefit the design and deployment of IoT application workflows. However, these frameworks are designed for expert users who have rich background knowledge.

Moreover, in order to reserve the correct workflow execution of IoT applications, a fault-tolerant system is necessary to overcome any faults caused by human activities and physical errors. Javed [123] proposed a fault-tolerant IoT architecture to offer failover of an interconnected network for both Edge and Cloud environments. Power [124] presented a microservices-based framework to detect fault-tolerance in real-time and predict fault patterns with machine learning mechanisms to mitigate faults before they activated. Grover [125] proposed an agent-based fault-tolerant IoT architecture to migrate data from the failed Edge servers to other available alternate

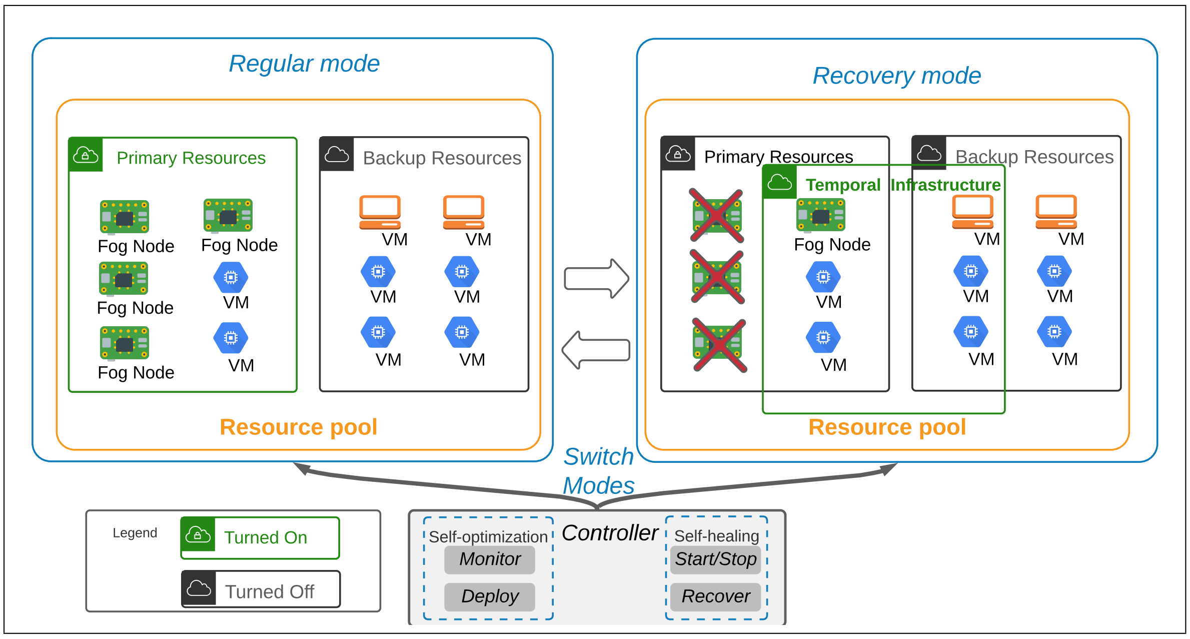


Figure 5.2: IoT Fault Recovery Model

servers to avoid system-level failure. However, these pieces of research fail to address multi-layer (Edge, Cloud) fault-tolerance interaction challenges, and they lack integrated deployment automation for failed service automatic redeploy.

5.3 IoT Fault-tolerance Model

This section presents a novel IoT fault-tolerance model and discusses details of the fault detection and recovery mechanism.

In our proposed IoT fault-tolerance model, shown in Figure 5.2, an IoT application consists of two main components: *controller* and *resource pool*. The controller is a fog node that controls the resource pool. It deploys the application, monitors resources, and switches to/from recovery mode. The controller manages all the available resources can be categorised into two resources: *primary resources* and *backup resources*. Primary resources refer to the resources utilised in a particular application as priority (usually considered edge resource and part of cloud resources). Backup resources represent resources for backup purpose (usually cloud resources). Since the available edge devices may have different architecture and configuration, it may not always be possible to have a backup device with the same configuration. Consequently, we choose on-demand cloud resources as the backup to meet the end-to-end latency constraints.

The system has two operational modes, *regular mode* and *recovery mode.* Regular mode is when the controller deploys IoT applications in the Primary Resources. The Recovery mode controller deploys in the *Temporal Infrastructure* (i.e., primary resources with chosen cloud backup resources) and recover failed resources. In the case of intolerable failure, the controller switches to recovery mode. After recovering failed nodes, it switches to the regular mode and resumes the deployment in the primary resources.

The centralised controller contains a Self-optimisation component and a Self-healing component. Self-optimisation manages continuous IoT applications deployments and detects an intolerable error in the primary resources group. Moreover, it operates in both primary resource (i.e., regular mode), and temporal infrastructure (i.e. recovery mode). Self-healing role can be summarised in three main actions: switch off failed resources, switch on backup resources and recover failed resources. Self-healing component prepares the temporal infrastructure for the Self-optimisation component by adding cloud resources to unfailing resources from primary resources. During recovery mode, the Self-healing component starting a set of recovery procedures including failed resources reboot, environment setup and availability check. After recovering failed nodes, the Self-optimisation component resumes the deployment in the regular mode. Self-healing reduces the cost of on-demand cloud VMs by switching off backup cloud resources during the regular mode.

With performing this novel IoT fault recovery model, users can efficiently execute a fault-tolerant IoT workflow application with a limited number of primary resources under a reasonable budget. Backup resources are considered when necessary and will be released immediately after recovered primary resources back to work. Such operations reduce the running cost and raise the reliability and availability of an IoT workflow application. As a result, we plan to develop an IoT workflow composition and deployment automation system, applying the proposed model to minimise application execution cost and maximise reliability. The details of this system are discussed in Section 5.4.

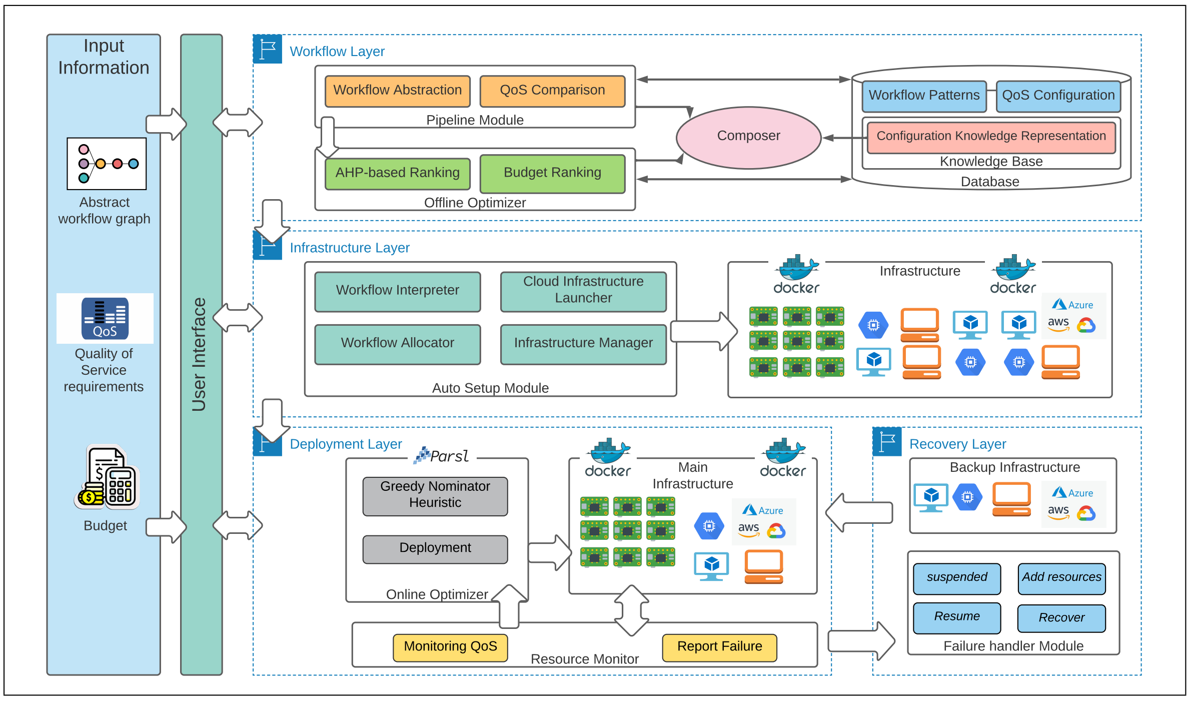


Figure 5.3: IoT Workflow Composition and Fault-tolerant System Architecture

5.4 System Design

This section presents the IoT workflow composition and fault-tolerant system architecture and discusses the system execution workflow.

5.4.1 System Architecture

According to the workflow composition and fault-tolerance challenges we discussed in Section 5.1, we propose an IoT workflow composition and fault-tolerant system to address these research problems. This system is designed to provide workflow composition and fault-tolerant workflow application execution which allows deployment automation and infrastructure recovery under proposed fault detection and recovery mechanism.

Figure 5.3 illustrates the layered schematic architecture of our proposed system and the dependencies of each component. This system is implemented as a web application that provides a *User Interface* for users to explore and compose their IoT components and execute fault-tolerant IoT workflow applications. Users can execute a fault-tolerant IoT application by simply inputting an abstract workflow graph, quality of service requirements and desired budget information. Other complex composition, deployment and recovery procedures are hidden to ease the interaction. There are four main layers of our system, as explained in the following subsections.

5.4.1.1 Workflow Layer

The workflow layer is proposed to manage IoT workflows. It consists of four main components, as discussed below:

Pipeline Module This component is involved in acquiring user input from the *User Interface* and creating a DAG-based IoT workflow. It has two modules *Workflow Abstraction* and *QoS Parameter Comparison*.

Workflow Abstraction relies on the abstract data analysis patterns available in the *workflow patterns* of the *QoS Parameter Comparison* allowing users to specify the priority of each non-functional QoS requirement stored in the *QoS Configuration* of the *Database*. The comparison values are required to be provided following the Saaty Scale[102]. A comparison matrix is constructed using these values and checked for consistency as the user entered values may not always be consistent. If the matrix is found to be inconsistent, a user is advised to enter new values.

Offline Optimizer Since there are many possible solutions for the deployment of each data analysis task, it is necessary to find an optimal solution which satisfies all the non-functional QoS requirements in the defined budget. However, finding an optimal solution for only one data analysis task can be proven to be NP-hard [126]. To solve this problem, we propose a heuristic model which is divided into two segments *AHP-based Ranking* and *Budget-based Ranking*.

AHP-based Ranking is based on the multi-criteria decision-making algorithm; Analytic Hierarchical Process (AHP) [102]. First, the algorithm takes as input the hierarchy of QoS requirements for each data analysis task and the QoS comparison matrix generated in the previous step. Using AHP, it then computes the priority of each QoS parameter and results in a weight vector. Finally, it takes the resource ranking values and computes the *Final Rank*, which is used to select the best infrastructure resource for the deployment.

Since some infrastructure resources may present a price which may not be suitable for a budget constraint user, *Budget Ranking* includes the infrastructure resource cost and total budget with the *Final Rank* given by *AHP-based Ranking*. This results in selecting an optimal resource for each data analysis task.

Database This is the most important component as it stores not only the basic *Workflow Patterns* and *QoS Configurations* but it also contains a *Knowledge Base*, which acts as a knowledge source for *Offline Optimizer* and *Composer*. *Knowledge Base* contains the predefined *Ranking* and *CKR* for different infrastructure resources. Both pieces of information are computed beforehand and available as ready to use. *Configuration Knowledge Representation (CKR)* is also computed for all the available resources using a knowledge management system *IoT-CANE* [127]. The scale of provided resources can be easily maintained using *IoT-CANE*.

Composer For each data analysis task, the *Composer* takes the optimized infrastructure component from *Offline Optimizer* and queries the *Database* for the desired *CKR*. Finally, it combines all the configurations and returns to the user in a unified format file which can be easily used for deployment purpose.

5.4.1.2 Infrastructure Layer

The infrastructure layer is designed to manage the setup of infrastructures used in IoT applications.

Auto Setup Module This module acquires composed workflow DAG with configuration information from the workflow layer. It manages infrastructure setup procedures including four components *Workflow Interpreter*, *Workflow Allocator*, *Cloud Infrastructure Launcher* and *Infrastructure Manager*.

Workflow Interpreter When a unified format IoT workflow file is composed and generated from *Composer*, this interpreter can read and understand the infrastructures required to deploy the particular IoT application. After interpreting, a list data structure containing each fog/cloud *CKR* is constructed for further deployment procedures. At the same time, *Workflow Allocator* is functioning to allocate and orchestrate such workflow activities to ensure the success of the workflow sequence.

As cloud providers, such as AWS, Azure and Google Cloud Provider offer different SDKs and APIs for developers to program and operate such cloud services, a unified, centralised *Cloud Infrastructure Launcher* becomes necessary for cloud environment setup. This component provides solutions for virtual machine launching, Docker installation, communication establishment, etc. Previous *CKR* information is adopted to specify the particular cloud provider, VM type, deployment location, network properties and other relevant requirements specified by APIs. When cloud infrastructures are ready to use, our system can receive the notification messages sent by these Docker installed VMs with static IP addresses for remote accessing.

After cloud infrastructures and fog/edge infrastructures are prepared, the *Infrastructure Manager* component offers CRUD (Create, Read, Update, Delete) operations for such infrastructures and attached containers. Because Docker is installed with REST API enabled on all launched infrastructures, the container management becomes comfortable with simple constructed RESTful requests. Meanwhile, to release the failed and unoccupied infrastructures is possible with simple commands from *Infrastructure Manager.*

In conclusion, the *Auto Setup Module* can interpret the composed workflow DAG and CKRs, then set up an infrastructure pool containing cloud/fog/edge infrastructures with Docker installed for further deployment processes.

5.4.1.3 Deployment Layer

The deployment layer is based on master-worker architecture. The master orchestrates the workflow and is located in the fog, whereas Workers are geo-distributed over fog-cloud infrastructure. Greed Nominator Heuristic (GNH) decided where to deploy the functions and utilises Parsl to control dataflow over the infrastructure. Parsl enables distributed parallel programming. The output of the infrastructure layer is divided into two infrastructures, primary and backup. In case where a failure event occurs the deployment layer notifies the recovery layer to prepare the backup node and recover the failures within the infrastructure.

The deployment layer includes the following parts:

Parsl controls computing geo-distributed resources that include managing connection, provisioning virtual resources to deploy DAG application. DataFlow Kernel (DFK) of Parsl handles error and steers dataflow between computing nodes.

The master node has the DFK and the configuration, as well as GNH to decided the redundant deployment. The master node connects, monitors resources and tracks failure. In a case where intolerable failure occurs, it generates the report with the failed nodes and the number of nodes to back up primary infrastructure.

Worker nodes are computing nodes that form the primary infrastructure. The virtual functions that are a part of the workflow application run on these worker nodes. Each virtual function is running within a docker container, which provides a flexible virtual function delivery. The master node does handling task allocation, where a worker provides its resources to compute it and pass the result to the master node or message broker (depends on the user-specification).

Online optimiser i.e., Greedy Nominator Heuristic (GNH), aims to i) lower end-to-end latency, ii) leverage redundancy to operate during failure events iii) Highest replicas to the critical function to lower risk of additional delay and iv) balancing redundancy and deployment cost. The greedy algorithm calculates the maximum possible replicas for each function using $MaxReplicas$ (formula 5.1) function, where i is the virtual function sequence in the application. The application size is represented by n, and m is a constant a user sets to adjust the redundancy number. Let $n = 5$, $m = 3$, formula 5.1 will produce a funnel-shaped redundant deployment similar to Figure 5.4

$$MaxReplicas = 1 + \lceil (1 - \frac{i}{n+1}) m \rceil \tag{5.1}$$

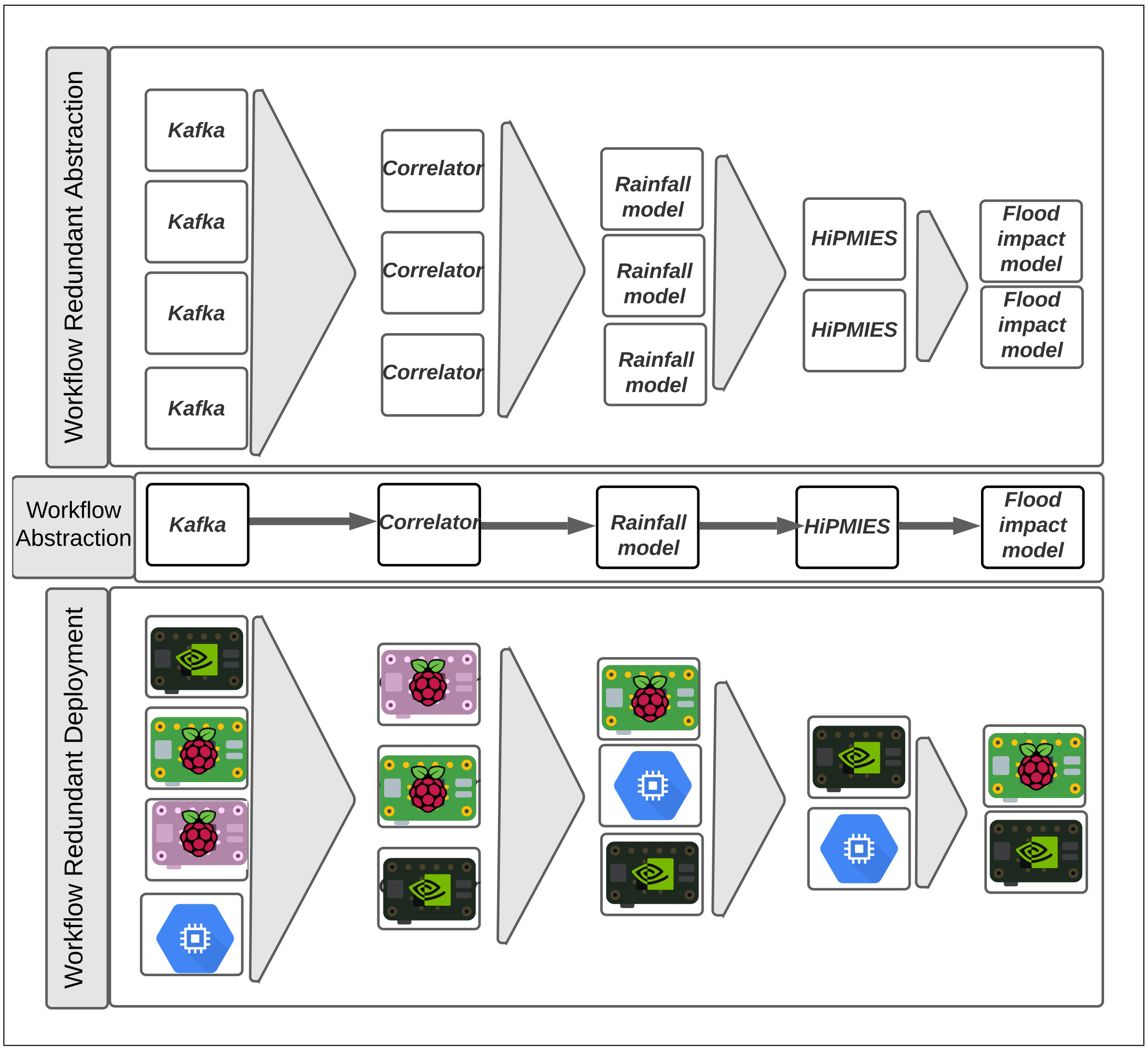


Figure 5.4: IoT Workflow Abstractions and Redundant Deployment

The GNH is scalable with the increasing number of nodes in the infrastructure. It searches in two stages i) Nomination phase and ii) Announcement phase. At the Nomination phase workers are divided between nominators, then each nominator provides partial-decision. The announcement phase decides the final redundant deployment out of the nominators' output. Redundant deployment is funnel-shaped, where earlier functions will have higher replicas.

5.4.1.4 Recovery Layer

The recovery layer is the layer that manages the backup infrastructure. It receives a failure report from the deployment layer then supports the central infrastructure with pre-configured cloud instance to form temporal infrastructure. While the temporal infrastructure receives tasks, the failed nodes are recovering. The backup nodes are turned off unless they are needed, this to reduce the operation cost.

Backup infrastructure is virtual machines in the cloud that has all the package dependency and the required configuration to run the IoT workflow. Each node in the backup infrastructure is mapped to one or two nodes in the primary infrastructure. This to ensure that there is a node that has the proper configuration to replace specific requirements.

Failure handle module is the component that controls the backup infrastructure and faulty nodes. The module can pause and resume back up nodes and recover the faulty node. Node failures mean that virtual instance is down, or its performance is deteriorating. Recovering a node is restoring the instance to its original configuration. This is achieved due to decoupling software from hardware. However, in case of a hardware failure, e.g., a malfunction within the electronic circuits, the services are moved from the faulty node to another backup node until the physical node is repaired/replaced and is ready for task execution.

DFK failure there are some occasions where the DFK fails to connect to workers. For example, in cases where the internet service providers assign a dynamic public IP to the master node, it will cause a disconnection to workers. In this case, the system has a standby Parsl's DFK that can resume task allocation to reduce whole system downtime. Figure 5.2 shows the recovery model with two DFK, where the red one is a paused DFK.

5.4.2 System Execution Workflow

The following subsection illustrates the main steps to execute IoT workflow composition and fault-tolerant system.

Define the IoT Workflow At the beginning, the *Workflow Layer* is designed to compose a different type of IoT application. The application is defined in terms of

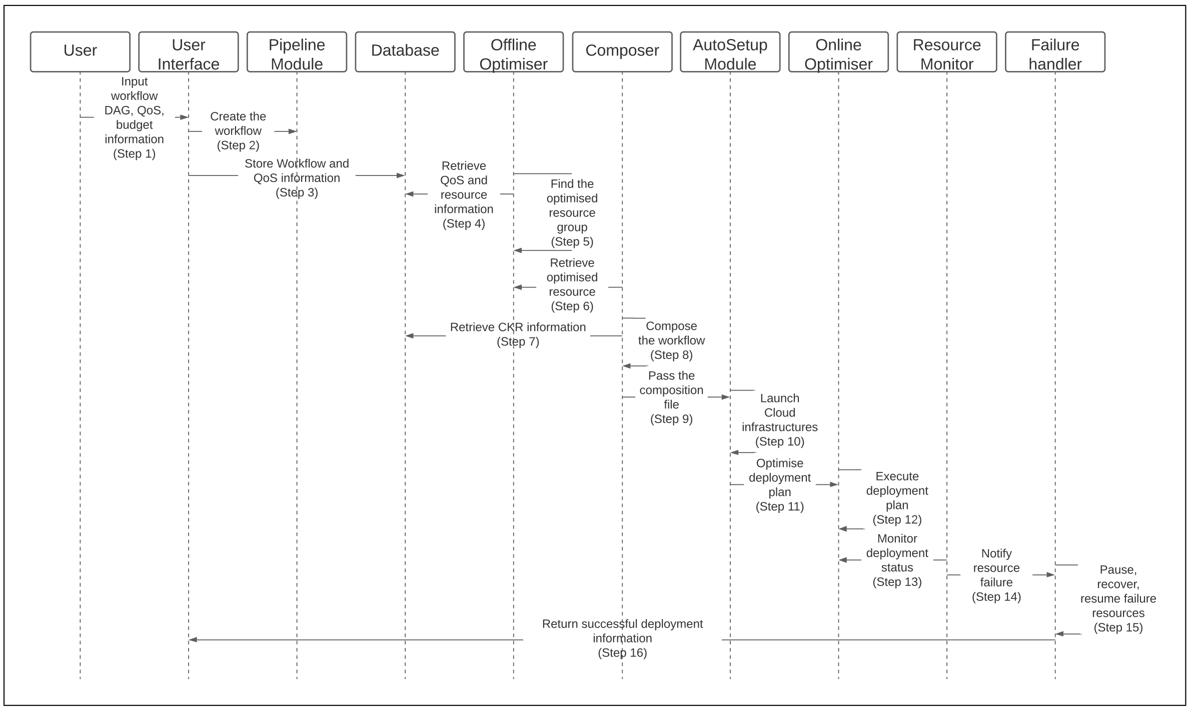

User
User Interface
Pipeline Module
Database
Offline Optimiser
Composer
AutoSetup Module
Online Optimiser
Resource Monitor
Failure handler
Input workflow DAG, QoS, budget information (Step 1)
Create the workflow (Step 2)
Store Workflow and QoS information (Step 3)
Retrieve QoS and resource information (Step 4)
Find the optimised resource group (Step 5)
Retrieve optimised resource (Step 6)
Retrieve CKR information (Step 7)
Compose the workflow (Step 8)
Pass the composition file (Step 9)
Launch Cloud infrastructures (Step 10)
Optimise deployment plan (Step 11)
Execute deployment plan (Step 12)
Monitor deployment status (Step 13)
Notify resource failure (Step 14)
Pause, recover, resume failure resources (Step 15)
Return successful deployment information (Step 16)

Figure 5.5: System Execution Workflow

data analysis and connection patterns using the *User Interface* as depicted in step 1 of Figure 5.5. Users can drag and drop the abstract patterns and rename them based on their adequacy. The pipeline module converts the user input into the workflow sequence, as shown in step 2, which is then stored in the database.

Define the QoS requirement comparison The *Workflow Layer* provides a two-way comparison scheme for all the QoS parameters. Users can enter a priority value in the box provided by *User Interface* which is converted into a comparison matrix by the *Pipeline Module*. Finally, the comparison matrix is stored in the database for further infrastructure ranking (Step 3).

Optimised infrastructure generation The *Offline Optimiser* retrieves the workflow and QoS and infrastructure information from the database (Step 4) and finds the optimised infrastructure for each workflow component (Step 5).

Compose the workflow The *Composer* retrieves the optimised infrastructure resource information from the *Offline Optimiser* (Step 6) and queries the *Knowledge*

Base to get the CKR information for the respective infrastructure resource (Step 7). Finally, it will compose the workflow (Step 8).

Infrastructure Setup The composed workflow information in JSON format can be transferred to *Workflow Interpreter* (Step 9) to interpret as a set of CKR information along with infrastructure locations, then pass them to *Workflow Allocator* for infrastructure orchestration. *Cloud Infrastructure Launcher* retrieves such information from the allocator and invokes API from cloud providers to launch necessary cloud resources with virtual programming environment installed (Step 10). Next, an infrastructure pool containing primary and backup resources is prepared for further deployment.

Workflow Deployment When the *Online Optimiser* receives a success message from infrastructure layer, a decision of virtual functions deployment allocation is generated by *Greedy Nominator Heuristic* module (Step 11). *Parsl* is enabled as a distributed parallel deployment tool to manage network connections and resource provisioning (Step 12). The DataFlow Kernel of Parsl handles errors and controls dataflow between computational resources.

Failure Handling During workflow application execution, *Resource Monitor* monitors and detects infrastructure level failure, then passes such failure information to *Failure Handler Module* (Steps 13, 14). The centralised controller can pause the failure resources first, then migrate tasks from failed nodes to backup resources to enable availability. Next, a recovery mechanism is employed to recover failed resources, then put them back to work (Step 15). Finally, successful deployment information is displayed on the user interface to indicate workflow deployment success (Step 16).

5.5 Evaluation

This section presents an IoT application utilised in our experiments and the related requirements following with experiment setup and failure model. We also explain and analyse the experiment results.

Table 5.1: Application Requirements

Sequences	Virtual Function	Task	Requirements	Processor
1	Kafka	Start Kafka server if not working	Internet access and privilege to open port 9092	CPU
2	Correlator	Check if there is a Correlator or start one	Access to the message broker	CPU
3	Rainfall model	Rainfall forecasting	Machine handles AI inference	CPU
4	HiPMIES	Predict sudden state changes during a flood event	Machine supports parallel computing	GPU
5	Flood impact model	Predict flood disruption at fixed locations	Machine handles AI inference	CPU

5.5.1 *Application Requirements*

This subsection describes the distributed application and the infrastructure requirements for the experimental application. We use real-time surface water flooding data monitoring and management application, called Flood-PREPARED [10], to evaluate our proposed system.

Any system issues such as component failure or late response will affect any adaptive response to a flood event. Thus, the system needs a fast response to the unreliability that can affect the Flood-PREPARED application's timing. The application is composed of containers and utilise Kafka streaming processing server to communicate messages. Parsl is the dataflow engine in this system that executes functions respecting data dependencies. The optimisation placement algorithm (i.e., GNH) decides the process nodes that deploy the containers. The application consists of the Rainfall model, HiPMIES (High-Performance Integrated hydrodynamic Modelling System) and Flood impact model. Application requirements details are shown in Table 5.1. Such models cooperate in monitoring and predicting flood in Newcastle city area.

5.5.2 *Experiment Setup*

The application runs over a Fog-Cloud environment. We utilise two cloud service provides: Amazon Web Service (AWS) and Google Cloud Platform (GCP). The infrastructure has three GPU nodes, one in the fog and two in GCP cloud. In total, the environment has ten computing nodes, half of it is primary, and the other half is backup nodes.

The HIPIMS requires GPU to execute CUDA program. Therefore, it is running in Nvidia Jetson nano or GPU instance with NVIDIA Tesla P100. All other functions are going to be running in every computing nodes. Controller and Kafka server is

Table 5.2: Infrastructure Computing Nodes

Qty	Computing Node	CPU	CPU speed	RAM	Storage	CUDA GPUs	Fog/Cloud zone	RTT
1	NVIDIA Jetson Nano	Quad core Cortex-A57 (ARM v8)	1.43 GHz	2 GB	32 GB	Maxwell	Fog	<1 ms
1	RPi 4 Model B	Quad core Cortex-A72 (ARM v8)	1.5 GHz	4 GB	32 GB	N/A	Fog	<1 ms
2	RPi 3 Model B+	Quad core Cortex-A53 (ARM v7)	1.4 GHz	1 GB	32 GB	N/A	Fog	<1 ms
1	RPi Zero W	Single core BCM2835 (ARM v6)	1 GHz	512 MB	32 GB	N/A	Fog	<1 ms
2	n1-standard-4 (GCP)	Dual core Intel Xeon - 4 vCPUs	2.30GHz	15 GB	50 GB	Tesla P100	europe-west1-b	99 ms
2	e2-medium (GCP)	Single core Intel Xeon - 2 vCPUs	2.30GHz	4 GB	50 GB	N/A	europe-west1-b	99 ms
2	t2.micro (AWS)	Single core Intel Xeon - 1 vCPUs	2.30GHz	1 GB	15 GB	N/A	eu-west-2a	93 ms

located in the fog infrastructure. The fog is located in Cardiff, the United Kingdom. The fog infrastructure contains Raspberry Pi ZeroW, Raspberry Pi 3B+, Raspberry Pi 4B and Nvidia Jetson Nano 2GB. Raspberry Pi 4B will be the controller and also will run some tasks in a virtual instance inside it. The CPU instances in GCP are e2-medium with 2 vCPUs, and 4 GB memory, whereas the GPU nodes are n1-standard-4 with 4 vCPUs and 15 GB memory). Each GCP has 50GB of storage. The GCP's nodes are located in Brussels, Belgium (europe-west1-b). The round trip time (RTT) of 14B package sent from the fog to GCP is 99 ms ($\pm$ 24.8). On the other hand, AWS instances are of t2.micro type, which has 1 vCPUs, and 1 GB memory. AWS instances have 15 GB of allocated disk. The AWS zone is eu-west-2 which located in London, the United Kingdom. The RTT of 14B packet between AWS and fog is 93 ms ($\pm$ 4.73). All detailed computing nodes information are shown in Table 5.2.

5.5.3 Failure Model

This subsection describes a time-dependent failure probability model from which we simulate node failure. Based on the *Weibull distribution*, we determined the probability of not completing a submitted task, i.e., deployed virtual function. The model parameters are as follows:

Start time is the timestamp when the node is started to deploy virtual functions, whether it is at the beginning or after a recovery session.

Current time is the count that started after the Start time represented by x, i.e., $x = \textit{current timestamp} - \textit{Start time}$.

Time-to-failure i.e., λ, is the time where the services in a node go down.

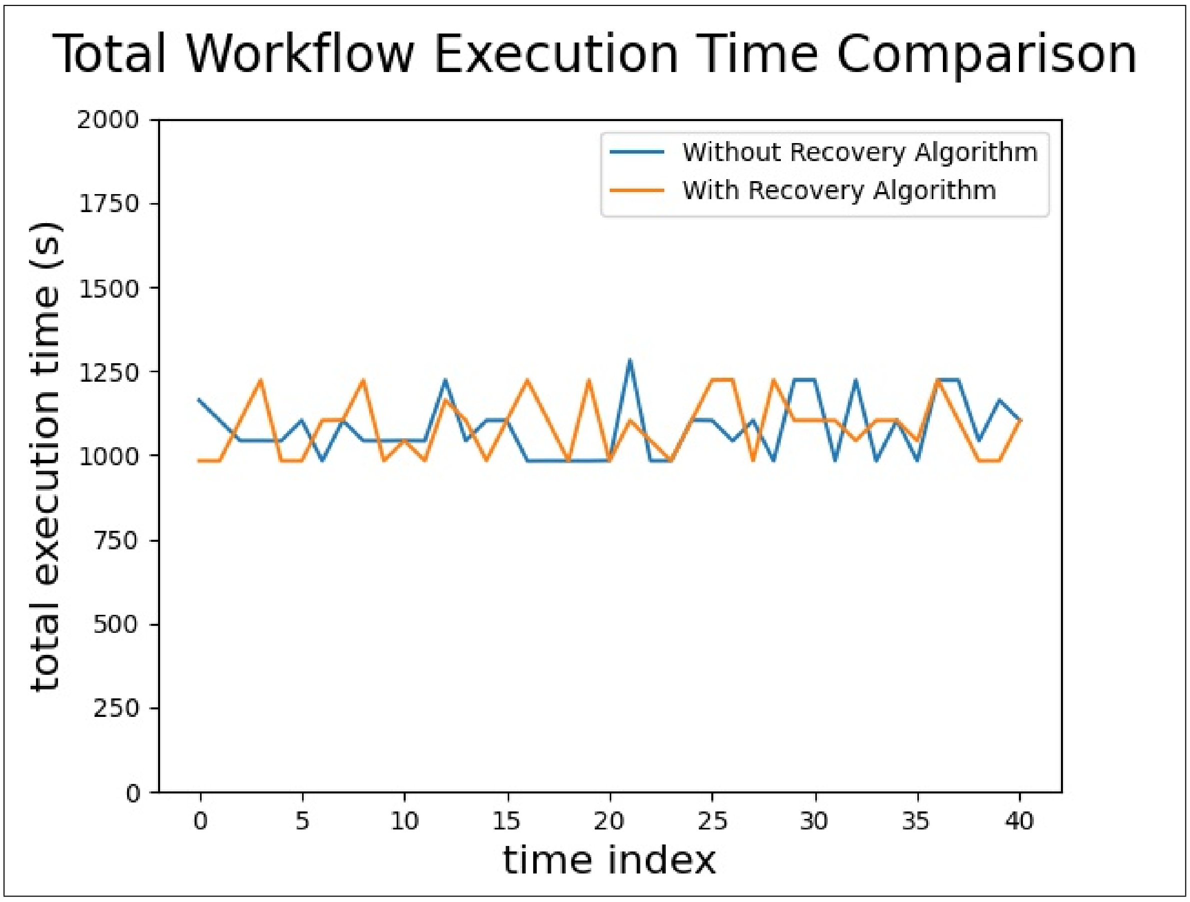


Figure 5.6: Total Execution Time Comparison

Reliability variable is Weibull shape parameter, i.e., k, that determines how reliable the node is. In case failure rate is constant then $k = 1$, whereas $k < 1$ or $k > 1$ means failure rate changes over time.

$$f(x; \lambda, k) = 1 - \mathrm{e}^{-(x/\lambda)^k} \tag{5.2}$$

A random choice based on probability of failure, i.e., $f(x; \lambda, k)$), and success, decides whether it met the deadline or not.

5.5.4 Experiment Result

This subsection describes the experimental results based on the proposed failure model and the Flood-PREPARED workflow.

In Table 5.3, we list resource parameters in terms of k and λ for both primary and backup resources. For the experiment parameters, we set a fixed λ (86400 sec) and

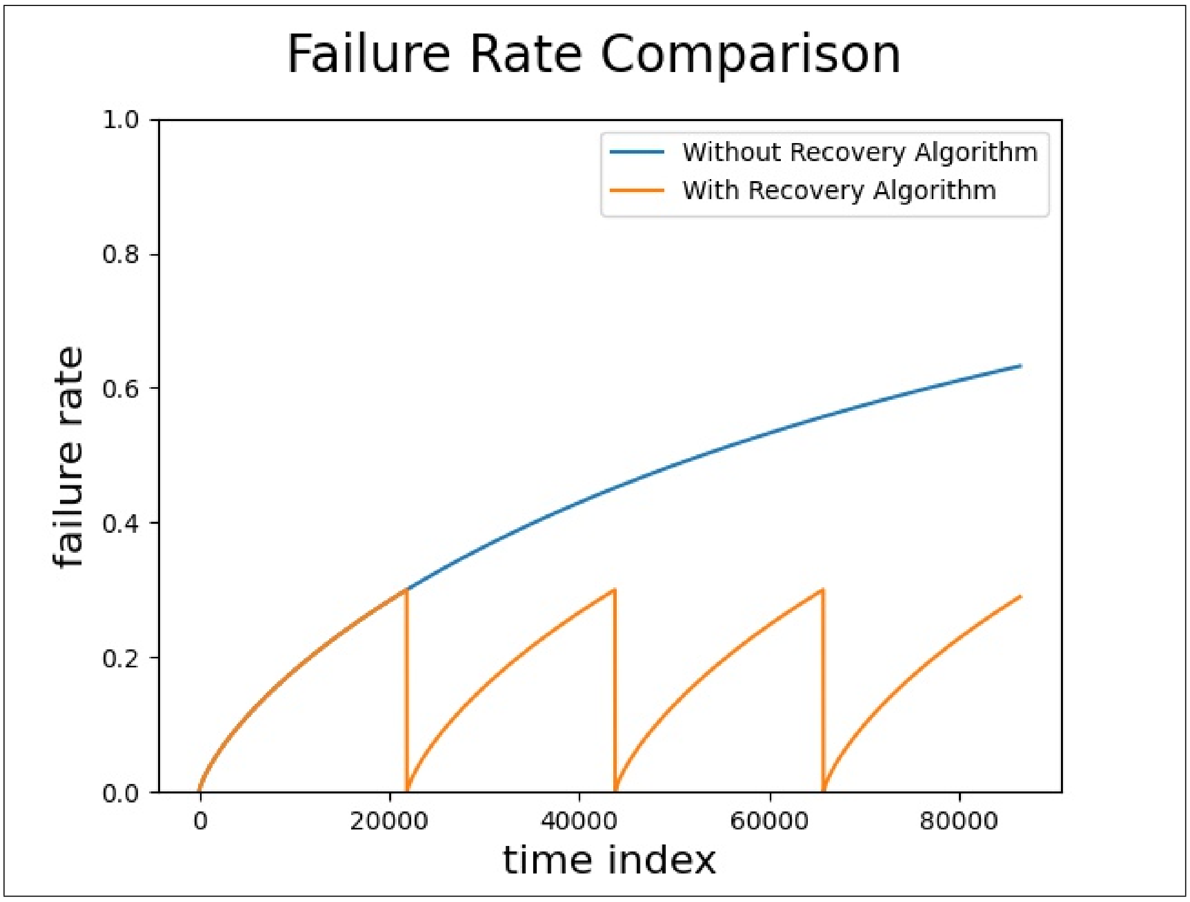


Figure 5.7: Failure Rate Comparison

Table 5.3: Experiment Parameters

parameter	Pi3 (No.1)	Pi3 (No.2)	AWS (backup)	Pi4	GCP	GCP (backup)	nano	GCP_GPU	GCP_GPU (backup)
k	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.9	0.9
lambda	86400	86400	86400	86400	86400	86400	86400	86400	86400

Table 5.4: Compare Cost

	AWS	***GCP***	***GCP_GPU***	***GCP_GPU***	***Total***
Price per Hour	$0.0116	$0.04	$2.3460	$2.3460	$4.7436
Uptime per day (minutes)	18.36	7.54	59.17	60.97	146.04
Bill with self-healing (month)	$0.11	$0.15	$69.40	$71.52	$141.18
Bill without self-healing (month)	$8.35	$28.80	$1,689.12	$1,689.12	$3,415.39
Decrease backup cost by	98.72%	99.48%	95.89%	95.77%	95.87%

vary k from 0.5 to 0.8 for different resources.

The final results of total workflow execution time and failure rate comparison in terms of recovery algorithm utilising are shown in Figure 5.6 and Figure 5.7 respectively.

We can clearly see from Figure 5.6 that the total workflow execution time has a rare difference if the recovery algorithm is enabled. Our recovery algorithm is designed to minimise the backup resources' cost while maintaining the total workflow execution time. Meanwhile, the failure rate increase overtime without the proposed recovery algorithm enabled. However, when the recovery algorithm is employed, the failure rate will drop after a time interval because the particular resource is in recovery processes.

Figure 5.7 shows that replicas of GNH guarantee task completed with no failure. Nevertheless, Table 5.4 illustrates the high cost of running extra cloud instances as backups (i.e., \$3,415.39 per month). Recovery strategies reduce the cost of on-demand backup nodes by 95.87% (i.e., \$141.18 per month) in total. Cost is reduced due to backups are shutdown unless they are needed.

5.6 Conclusion

This chapter has described a novel framework which is utilised to compose IoT workflow from given DAG and QoS requirements, then automatically launch cloud infrastructures with recommended configurations, finally allocate and deploy desired virtual functions across edge and cloud environments. Meanwhile, the proposed system detects and recovers infrastructure level failures to enable continuous services with tolerable end-to-end latency. This system has been validated and evaluated with a real-world surface water flooding data monitoring and management application. The results of experiments proves efficiency and effectiveness of our system. In the future work, an execution feedback-based intelligent infrastructure recommendation approach can be employed to increase the configuration recommendation accuracy and decrease useless budget waste.

6

Conclusions and Future Work

Contents

Summary

This chapter summarise the research work presented in this thesis. I highlight the contributions of the thesis and discuss the research directions about the future work.

6.1 Thesis Summary

With the growth of IoT paradigm, it is of paramount importance to provide simplified workflow and deployment solutions. In this thesis, I designed and developed a set of frameworks to enable unified workflow composition and fault-tolerant automation deployment in multi-cloud and edge environments in order to meet demands and QoS constraints.

Chapter 2 First, I gave an overview of background information about IoT. Then the dimensions of configuration management and automation deployment were classified into ten categories: Dependency graph, Access mechanism, Access control, Extensibility, Customisation, Reusability, Deployment environment, Virtualisation technique, Scalability and Portability, while the state of the art of configuration management and automation deployment tools were evaluated. Next, I investigated the taxonomy of scientific workflow system and the difference with IoT workflow. Finally, I discussed IoT workflow composition and orchestration objectives and current available tools.

Chapter 3 I proposed a unified conceptual model which captures the resource configurations in IoT environments. Besides, a support recommendation system for the resource configuration recommendation in IoT utilising SQL-based relational semantics and procedures was presented. Then, I developed an incremental method to facilitate the knowledge acquisition in IoT resource configuration knowledge base. Finally, I designed a service interface that converts simple context information captured from users to optimal IoT resource configurations to map users' requirements. Meanwhile, I validated and evaluated the proposed framework with an implementation and a user study.

Chapter 4 I proposed and developed a novel composition framework, IoT workflow composition system (IoTWC). I provide basic IoT workflow activity patterns that are abstracted within IoTWC based on literature study and interviewing of domain experts. IoTWC leverages the analytic hierarchy process (AHP) to compose the multi-level IoT workflow that satisfy the requirements of any IoT application. Then I validated the coverage of IoT workflow activity abstract patterns with a user case study. I also evaluated IoTWC with a real-world scenario for smart buildings which show the effectiveness of the proposed system in terms of IoT workflow abstraction and composition.

Chapter 5 I proposed a framework that enables a user to compose any IoT application with defined QoS parameters, then automatically set up the cloud infrastructures according to the recommended configurations. The system is divided into two components, Self-configuration and Self-optimisation. Self-configuration takes users QoS requirements and a workflow DAG of an IoT application; then it returns ready to user infrastructure group. Self-optimisation component continuously adapted to traffic and the infrastructure state to execute the IoT application based on a novel fault-tolerance model. Finally, the proposed framework has been validated and evaluated with a real-world surface water flooding data monitoring and management application, where the results proved the efficiency and effectiveness.

In conclusion, these proposed frameworks and systems can empower and simplify IoT application from design time to deploy period. More specifically, with IoT-CANE enabled, engineers from Flood-PREPARED can maintain optimal IoT resource configuration knowledge artifects by delivering simple context information to this system. Meanwhile, these configuration knowledge artifects can be composed to proper IoT workflow DAG by IoTWC, to offer ranked IoT application deployment solutions. With the proposed fault-tolerant IoT system, this Flood-PREPARED can be deployed automatically with given configurations. Both edge and cloud resources in Flood-PREPARED are also fault-tolerant with proposed infrastructure recovery algorithm.

6.2 Future Research Directions

Alongside my contributions, I also provide a number of opportunities of future research.

6.2.1 Dynamic Distributed Workflow Deployment

There are a large number of IoT workflow systems designed to bring centralised IoT workflow deployment. However, the majority of workflow systems are designed to operate in a fixed networked environment, such as cloud environment or private network. Such systems rely on a central point of coordination in order to manage centralised workflow deployment. A dynamic distributed workflow deployment mechanism which leverages a decentralised execution approach to run on coordinative edge and cloud network environment is a new challenge.

6.2.2 Resource Prediction in IoT Workflow Composition

The current workflow orchestration strategies focus on workflow tasks deployment on the collaborative edge and cloud environment without concerning the availability and current load of the resources. However, the resources selection based on the dynamic QoS constraints is one of the largest research problems in workflow orchestration. To overcome previous research challenges, a resource prediction approach is needed. The resource prediction mechanism may help to allocate the suitable running nodes for a workflow based on the resource availability and the runtime loads stats which may increase the accuracy of the application performance.

6.2.3 Feedback-based IoT Workflow Composition

When performing a workflow composition procedure, a configuration knowledge base is utilised to provide historical data. Meanwhile, several resource allocation strategies help to recommend suitable resources through edge and cloud environments. However, due to the dynamic QoS requirements, historical configuration data may not able to produce accuracy results. As a result, it is challenging to deliver a feedback-based intelligent IoT workflow composition framework. The feedback-based framework can monitor and collect deployment performance feedback, and allocate future workflow tasks based on these performance feedback analysis proposed by machine learning algorithms.

6.2.4 Workload Prediction based Workflow Orchestration

The collaborative edge and cloud environment consists of broad heterogeneous resources which are QoS constraints sensitive. It is a crucial task to understand the patterns and characteristics of workload in order to improve the system operational conditions and resource utilisation. Workload analysis and target server property prediction based on realistic parameters such as memory and CPU usage are required to explore the federated workload characteristics. Performance prediction model helps to discover the best-fit target resource more accurately. The merit of the performance model depends on the historical resource usage information like CPU and memory usage, resources performance rate, the workflow execution time, etc.

Bibliography

[1] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of things (iot): A literature review," *Journal of Computer and Communications*, vol. 3, no. 05, p. 164, 2015.

[4] Y. Tang, D. Chen, L. Wang, A. Y. Zomaya, J. Chen, and H. Liu, "Bayesian tensor factorization for multi-way analysis of multi-dimensional eeg," *Neurocomputing*, vol. 318, pp. 162–174, 2018.

[5] D. Chen, Y. Hu, L. Wang, A. Y. Zomaya, and X. Li, "H-parafac: Hierarchical parallel factor analysis of multidimensional big data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1091–1104, 2017.

[6] D. Chen, X. Li, L. Wang, S. U. Khan, J. Wang, K. Zeng, and C. Cai, "Fast and scalable multi-way analysis of massive neural data," *IEEE Transactions on Computers*, vol. 64, no. 3, pp. 707–719, 2015.

[7] H. Ke, D. Chen, T. Shah, X. Liu, X. Zhang, L. Zhang, and X. Li, "Cloud-aided online eeg classification system for brain healthcare: A case study of depression evaluation with a lightweight cnn," *Software: Practice and Experience*, 2018.

[8] J. Fan, J. Yan, Y. Ma, and L. Wang, "Big data integration in remote sensing across a distributed metadata-based spatial infrastructure," *Remote Sensing*, vol. 10, no. 1, p. 7, 2017.

[9] M. Blackstock and R. Lea, "Iot interoperability: A hub-based approach," in *Internet of Things (IOT), 2014 International Conference on the*, pp. 79–84, IEEE, 2014.

[10] S. Barr, S. Johnson, X. Ming, M. Peppa, N. Dong, Z. Wen, C. Robson, L. Smith, P. James, D. Wilkinson, *et al.*, "Flood-prepared: A nowcasting system for real-time impact adaption to surface water flooding in cities," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 6, pp. 9–15, 2020.

[11] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[12] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International journal of communication systems*, vol. 25, no. 9, p. 1101, 2012.

[13] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[14] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

[15] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[16] M. Chen, J. Wan, and F. Li, "Machine-to-machine communications: Architectures, standards and applications.," *Ksii transactions on internet & information systems*, vol. 6, no. 2, 2012.

[17] K. Su, J. Li, and H. Fu, "Smart city and the applications," in *2011 international conference on electronics, communications and control (ICECC)*, pp. 1028–1031, IEEE, 2011.

[18] S. Mittal and A. Tolk, *Complexity Challenges in Cyber Physical Systems: Using Modeling and Simulation (M&S) to Support Intelligence, Adaptation and Autonomy*. John Wiley & Sons, 2019.

[19] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[20] Y. Xing and Y. Zhan, "Virtualization and cloud computing," in *Future Wireless Networks and Information Systems*, pp. 305–312, Springer, 2012.

[21] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pp. 171–172, IEEE, 2015.

[22] J. Zhang, H. Huang, and X. Wang, "Resource provision algorithms in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 64, pp. 23–42, 2016.

[23] L. Carnevale, A. Celesti, A. Galletta, S. Dustdar, and M. Villari, "From the cloud to edge and iot: a smart orchestration architecture for enabling osmotic computing," in *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 419–424, IEEE, 2018.

[24] G. Kecskemeti, G. Casale, D. N. Jha, J. Lyon, and R. Ranjan, "Modelling and simulation challenges in internet of things," *IEEE cloud computing*, vol. 4, no. 1, pp. 62–69, 2017.

[25] H. B. Pötter and A. Sztajnberg, "Adapting heterogeneous devices into an iot context-aware infrastructure," in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 64–74, 2016.

[26] C. Lueninghoener, "Getting started with configuration management," *USENIX; login*, vol. 36, no. 2, pp. 12–17, 2011.

[27] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "Tosca: portable automated deployment and management of cloud applications," in *Advanced Web Services*, pp. 527–549, Springer, 2014.

[28] A. C. F. da Silva, U. Breitenbücher, K. Képes, O. Kopp, and F. Leymann, "Opentosca for iot: automating the deployment of iot applications based on the mosquitto message broker," in *Proceedings of the 6th International Conference on the Internet of Things*, pp. 181–182, ACM, 2016.

[29] A. Mehta, R. Baddour, F. Svensson, H. Gustafsson, and E. Elmroth, "Calvin constrained—a framework for iot applications in heterogeneous environments," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1063–1073, IEEE, 2017.

[30] P. Persson and O. Angelsmark, "Calvin–merging cloud and iot," *Procedia Computer Science*, vol. 52, pp. 210–217, 2015.

[31] W. Van Der Aalst, K. M. Van Hee, and K. van Hee, *Workflow management: models, methods, and systems.* MIT press, 2004.

[32] N. Russell, A. H. Ter Hofstede, D. Edmond, and W. M. van der Aalst, "Workflow data patterns: Identification, representation and tool support," in *International Conference on Conceptual Modeling*, pp. 353–368, Springer, 2005.

[33] A. Barker and J. Van Hemert, "Scientific workflow: a survey and research directions," in *International Conference on Parallel Processing and Applied Mathematics*, pp. 746–753, Springer, 2007.

[34] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future generation computer systems*, vol. 25, no. 5, pp. 528–540, 2009.

[35] F.-S. Hsieh and J.-B. Lin, "A self-adaptation scheme for workflow management in multi-agent systems," *Journal of Intelligent Manufacturing*, vol. 27, no. 1, pp. 131–148, 2016.

[36] W. Viriyasitavat, *A framework of trust in service workflows.* PhD thesis, University of Oxford, 2013.

[37] S. Feja, S. Witt, and A. Speck, "Bam: A requirements validation and verification framework for business process models," in *2011 11th International Conference on Quality Software*, pp. 186–191, IEEE, 2011.

[38] S. Feja and D. Fötsch, "Model checking with graphical validation rules," in *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*, pp. 117–125, IEEE, 2008.

[39] H. Davulcu, M. Kifer, L. R. Pokorny, C. Ramakrishnan, I. Ramakrishnan, and S. Dawson, "Modeling and analysis of interactions in virtual enterprises," in *Proceedings Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises. RIDE-VE'99*, pp. 12–18, IEEE, 1999.

[40] D. Bianculli, C. Ghezzi, and P. San Pietro, "The tale of soloist: a specification language for service compositions interactions," in *International Workshop on Formal Aspects of Component Software*, pp. 55–72, Springer, 2012.

[41] W. Viriyasitavat, L. Da Xu, and A. Martin, "Swspec: The requirements specification language in service workflow environments," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 3, pp. 631–638, 2012.

[42] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, *et al.*, "Business process execution language for web services," 2003.

[43] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, *et al.*, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[44] C. Team, "Dagman: A directed acyclic graph manager," *See website at http://www. cs. wisc. edu/condor/dagman*, 2005.

[45] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[46] H. D. Lord, "Improving the application development process with modular visualization environments," *ACM Siggraph Computer Graphics*, vol. 29, no. 2, pp. 10–12, 1995.

[47] S. G. Parker, M. Miller, C. D. Hansen, and C. R. Johnson, "An integrated problem solving environment: the scirun computational steering system," in *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, vol. 7, pp. 147–156, IEEE, 1998.

[48] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," in *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pp. 423–424, IEEE, 2004.

[49] I. Taylor, M. Shields, I. Wang, and A. Harrison, "Visual grid workflow in triana," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 153–169, 2005.

[50] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Managing the evolution of dataflows with vistrails," in *22nd International Conference on Data Engineering Workshops (ICDEW'06)*, pp. 71–71, IEEE, 2006.

[51] Y. Gil, "Workflow composition: Semantic representations for flexible automation," in *Workflows for e-Science*, pp. 244–257, Springer, 2007.

[52] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim, "Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 22, p. 1767, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[53] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures*, vol. 7. University of California, Irvine Irvine, 2000.

[54] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, *et al.*, "Web services description language (wsdl) 1.1," 2001.

[55] A. Kertész, G. Sipos, and P. Kacsuk, "Brokering multi-grid workflows in the p-grade portal," in *European Conference on Parallel Processing*, pp. 138–149, Springer, 2006.

[56] T. Delaitre, T. Kiss, A. Goyeneche, G. Terstyanszky, S. Winter, and P. Kacsuk, "Gemlca: Running legacy code applications as grid services," *Journal of Grid Computing*, vol. 3, no. 1-2, pp. 75–90, 2005.

[57] G. Allen, T. Goodale, T. Radke, M. Russell, E. Seidel, K. Davis, K. N. Dolkas, N. D. Doulamis, T. Kielmann, A. Merzky, *et al.*, "Enabling applications on the grid: A gridlab overview," *The International Journal of High Performance Computing Applications*, vol. 17, no. 4, pp. 449–466, 2003.

[58] I. Taylor, M. Shields, I. Wang, and O. Rana, "Triana applications within grid computing and peer to peer environments," *Journal of Grid Computing*, vol. 1, no. 2, pp. 199–217, 2003.

[59] G. Von Laszewski, M. Hategan, and D. Kodeboyina, "Java cog kit workflow," in *Workflows for e-Science*, pp. 340–356, Springer, 2007.

[60] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[61] R. Henderson and D. Tweten, "Portable batch system: External reference specification," tech. rep., Technical report, NASA, Ames Research Center, 1996.

[62] S. Zhou, "Lsf: Load sharing in large heterogeneous distributed systems," in *I Workshop on cluster computing*, vol. 136, 1992.

[63] A. J. Younge, G. Von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers, "Efficient resource management for cloud computing environments," in *International Conference on Green Computing*, pp. 357–364, IEEE, 2010.

[64] V. Welch, F. Siebenlist, I. Foster12, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Gsi3: Security for grid services," in *Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing (HPDC-12)*, Citeseer, 2003.

[65] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf, “The cactus worm: Experiments with dynamic resource discovery and allocation in a grid environment,” *The International Journal of High Performance Computing Applications*, vol. 15, no. 4, pp. 345–358, 2001.

[66] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo, “Vistrails: visualization meets data management,” in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 745–747, 2006.

[67] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan, “Bi-criteria workflow tasks allocation and scheduling in cloud computing environments,” in *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 638–645, IEEE, 2012.

[68] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[69] M. Nardelli, S. Nastic, S. Dustdar, M. Villari, and R. Ranjan, “Osmotic flow: Osmotic computing+ iot workflow,” *IEEE Cloud Computing*, vol. 4, no. 2, pp. 68–75, 2017.

[70] M. Adhikari, T. Amgoth, and S. N. Srirama, “A survey on scheduling strategies for workflows in cloud environment and emerging trends,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–36, 2019.

[71] R. Khorsand, F. Safi-Esfahani, N. Nematbakhsh, and M. Mohsenzade, “Taxonomy of workflow partitioning problems and methods in distributed environments,” *Journal of Systems and Software*, vol. 132, pp. 253–271, 2017.

[72] M. Masdari, S. ValiKardan, Z. Shahi, and S. I. Azar, “Towards workflow scheduling in cloud computing: a comprehensive analysis,” *Journal of Network and Computer Applications*, vol. 66, pp. 64–82, 2016.

[73] D. Poola, M. A. Salehi, K. Ramamohanarao, and R. Buyya, “A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments,” in *Software architecture for big data and the cloud*, pp. 285–320, Elsevier, 2017.

[74] S. Smanchat and K. Viriyapant, “Taxonomies of workflow scheduling problem and techniques in the cloud,” *Future Generation Computer Systems*, vol. 52, pp. 1–12, 2015.

[75] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, “Cost efficient resource management in fog computing supported medical cyber-physical system,” *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 1, pp. 108–119, 2015.

[76] L. Peng, A. R. Dhaini, and P.-H. Ho, "Toward integrated cloud–fog networks for efficient iot provisioning: Key challenges and solutions," *Future Generation Computer Systems*, vol. 88, pp. 606–613, 2018.

[77] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, pp. 1–43, 2017.

[78] P. Ferrari, E. Sisinni, D. Brandão, and M. Rocha, "Evaluation of communication latency in industrial iot applications," in *2017 IEEE International Workshop on Measurement and Networking (M&N)*, pp. 1–6, IEEE, 2017.

[79] F. Tao, Y. Wang, Y. Zuo, H. Yang, and M. Zhang, "Internet of things in product life-cycle energy management," *Journal of Industrial Information Integration*, vol. 1, pp. 26–39, 2016.

[80] N. Kaur and S. K. Sood, "An energy-efficient architecture for the internet of things (iot)," *IEEE Systems Journal*, vol. 11, no. 2, pp. 796–805, 2015.

[81] S. K. Datta, R. P. F. Da Costa, and C. Bonnet, "Resource discovery in internet of things: Current trends and future standardization aspects," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pp. 542–547, IEEE, 2015.

[82] S. De, P. Barnaghi, M. Bauer, and S. Meissner, "Service modelling for the internet of things," in *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pp. 949–955, IEEE, 2011.

[83] M. Bauer, N. Bui, J. De Loof, C. Magerkurth, A. Nettsträter, J. Stefa, and J. W. Walewski, "Iot reference model," in *Enabling Things to Talk*, pp. 113–162, Springer, 2013.

[84] W. Wang, S. De, G. Cassar, and K. Moessner, "Knowledge representation in the internet of things: semantic modelling and its applications," *automatika*, vol. 54, no. 4, pp. 388–400, 2013.

[85] H. Neuhaus and M. Compton, "The semantic sensor network ontology," in *AGILE workshop on challenges in geospatial data harmonisation, Hannover, Germany*, pp. 1–33, 2009.

[86] W.-P. Lee, C. Kaoli, and J.-Y. Huang, "A smart tv system with body-gesture control, tag-based rating and context-aware recommendation," *Knowledge-Based Systems*, vol. 56, pp. 167–178, 2014.

[87] D. Weerasiri and B. Benatallah, "Unified representation and reuse of federated cloud resources configuration knowledge," in *Enterprise Distributed Object Computing Conference (EDOC), 2015 IEEE 19th International*, pp. 142–150, IEEE, 2015.

[88] M. Zhang, R. Ranjan, M. Menzel, S. Nepal, P. Strazdins, W. Jie, and L. Wang, "An infrastructure service recommendation system for cloud applications with real-timeqosrequirement constraints," *IEEE Systems Journal*, 2015.

[89] B. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pp. 85–90, IEEE, 1994.

[90] Z.-L. Chen, S. Raghavan, P. Gray, and H. J. Greenberg, “State-of-the-art decision-making tools in the information-intensive age,” 2008.

[91] B. R. Gaines and P. Compton, “Induction of ripple-down rules applied to modeling large databases,” *Journal of Intelligent Information Systems*, vol. 5, no. 3, pp. 211–228, 1995.

[92] B. Kang, P. Compton, and P. Preston, “Multiple classification ripple down rules: evaluation and possibilities,” in *Proceedings 9th Banff knowledge acquisition for knowledge based systems workshop*, vol. 1, pp. 17–1, 1995.

[93] M. Kranz, P. Holleis, and A. Schmidt, “Embedded interaction: Interacting with the internet of things,” *IEEE internet computing*, no. 2, pp. 46–53, 2009.

[94] D. N. Jha, P. Michalak, Z. Wen, P. Watson, and R. Ranjan, “Multi-objective deployment of data analysis operations in heterogeneous iot infrastructure,” *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2019.

[95] M. Dimitrov, A. Simov, S. Stein, and M. Konstantinov, “A bpmo based semantic business process modelling environment,” in *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM-2007)*, vol. 251, pp. 1613–0073, 2007.

[96] D. Chiu, T. Hall, F. Kabir, and G. Agrawal, “An approach towards automatic workflow composition through information retrieval,” in *Proceedings of the 15th Symposium on International Database Engineering & Applications*, pp. 170–178, ACM, 2011.

[97] P. Asghari, A. M. Rahmani, and H. Haj Seyyed Javadi, “A medical monitoring scheme and health-medical service composition model in cloud-based iot platform,” *Transactions on Emerging Telecommunications Technologies*, p. e3637.

[98] P. Partheeban and V. Kavitha, “Versatile provisioning and workflow scheduling in waas under cost and deadline constraints for cloud computing,” *Transactions on Emerging Telecommunications Technologies*, vol. 30, no. 1, p. e3527, 2019.

[99] G. Nikol, M. Träger, S. Harrer, and G. Wirtz, “Service-oriented multi-tenancy (so-mt): enabling multi-tenancy for existing service composition engines with docker,” in *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pp. 238–243, IEEE, 2016.

[100] K. Hightower, B. Burns, and J. Beda, *Kubernetes: up and running: dive into the future of infrastructure.* ” O’Reilly Media, Inc.”, 2017.

[101] A. Amazon, “Cloud formation,” *Amazon Web Services. AWS CloudFormation. Available online: http://aws. amazon. com/de/cloudformation.*

[102] T. L. Saaty, "Analytic hierarchy process," *Encyclopedia of Biostatistics*, vol. 1, 2005.

[103] N. Glombitza, S. Ebers, D. Pfisterer, and S. Fischer, "Using bpel to realize business processes for an internet of things," in *International Conference on Ad-Hoc Networks and Wireless*, pp. 294–307, Springer, 2011.

[104] D. Domingos, F. Martins, C. Cândido, and R. Martinho, "Internet of things aware ws-bpel business processes context variables and expected exceptions.," *J. UCS*, vol. 20, no. 8, pp. 1109–1129, 2014.

[105] G. Chen, J. Huang, B. Cheng, and J. Chen, "A social network based approach for iot device management and service composition," in *2015 IEEE World Congress on Services*, pp. 1–8, IEEE, 2015.

[106] A. Urbieta, A. González-Beltrán, S. B. Mokhtar, M. A. Hossain, and L. Capra, "Adaptive and context-aware service composition for iot-based smart cities," *Future Generation Computer Systems*, vol. 76, pp. 262–274, 2017.

[107] R. Chen, J. Guo, and F. Bao, "Trust management for soa-based iot and its application to service composition," *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 482–495, 2014.

[108] T. Baker, M. Asim, H. Tawfik, B. Aldawsari, and R. Buyya, "An energy-aware service composition algorithm for multiple cloud-based iot applications," *Journal of Network and Computer Applications*, vol. 89, pp. 96–108, 2017.

[109] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE transactions on services Computing*, vol. 5, no. 2, pp. 164–177, 2011.

[110] S. A. Karthikeya, J. Vijeth, and C. S. R. Murthy, "Leveraging solution-specific gateways for cost-effective and fault-tolerant iot networking," in *2016 IEEE Wireless Communications and Networking Conference*, pp. 1–6, IEEE, 2016.

[111] H. Madsen, B. Burtschy, G. Albeanu, and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable fog computing," in *2013 20th International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 43–46, IEEE, 2013.

[112] M. Zorzi, A. Gluhak, S. Lange, and A. Bassi, "From today's intranet of things to a future internet of things: a wireless-and mobility-related view," *IEEE Wireless communications*, vol. 17, no. 6, pp. 44–51, 2010.

[113] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad hoc networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[114] D. Zhang, L. T. Yang, and H. Huang, "Searching in internet of things: Vision and challenges," in *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*, pp. 201–206, IEEE, 2011.

[115] X. Yu, B. Nguyen, and Y. Chen, "Internet of things capability and alliance: Entrepreneurial orientation, market orientation and product and process innovation," *Internet Research*, vol. 26, no. 2, pp. 402–434, 2016.

[116] S. C. Mukhopadhyay and N. K. Suryadevara, "Internet of things: Challenges and opportunities," in *Internet of Things*, pp. 1–17, Springer, 2014.

[117] C. Perera, R. Ranjan, L. Wang, S. U. Khan, and A. Y. Zomaya, "Big data privacy in the internet of things era," *IT Professional*, vol. 17, no. 3, pp. 32–39, 2015.

[118] R. Roman, P. Najera, and J. Lopez, "Securing the internet of things," *Computer*, vol. 44, no. 9, pp. 51–58, 2011.

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

[120] A. Kumar, N. C. Narendra, and U. Bellur, "Uploading and replicating internet of things (iot) data on distributed cloud storage," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pp. 670–677, IEEE, 2016.

[121] M. A. Hail, M. Amadeo, A. Molinaro, and S. Fischer, "Caching in named data networking for the wireless internet of things," in *2015 international conference on recent advances in internet of things (RIoT)*, pp. 1–6, IEEE, 2015.

[122] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Mobile edge computing and networking for green and low-latency internet of things," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 39–45, 2018.

[123] A. Javed, K. Heljanko, A. Buda, and K. Främling, "Cefiot: A fault-tolerant iot architecture for edge and cloud," in *2018 IEEE 4th world forum on internet of things (WF-IoT)*, pp. 813–818, IEEE, 2018.

[124] A. Power and G. Kotonya, "A microservices architecture for reactive and proactive fault tolerance in iot systems," in *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pp. 588–599, IEEE, 2018.

[125] J. Grover and R. M. Garimella, "Reliable and fault-tolerant iot-edge architecture," in *2018 IEEE SENSORS*, pp. 1–4, IEEE, 2018.

[126] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[127] Y. Li, A. Alqahtani, E. Solaiman, C. Perera, P. P. Jayaraman, R. Buyya, G. Morgan, and R. Ranjan, "Iot-cane: A unified knowledge management system for data-centric internet of things application systems," *Journal of Parallel and Distributed Computing*, vol. 131, pp. 161–172, 2019.