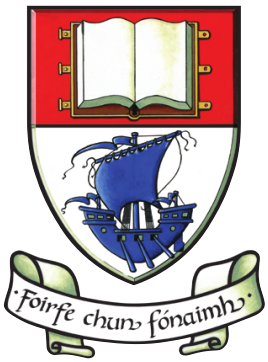
Fog Computing Support for Internet of Things Applications



Mohit Taneja

Department of Computing and Mathematics
School of Science and Computing

Waterford Institute of Technology

This dissertation is submitted for the degree of

Doctor of Philosophy

Supervisor: Dr. Alan Davy

Submitted to Waterford Institute of Technology, September 2020

This thesis is dedicated to my *family*— not just the bonds connected by blood, but the people who stay around when needed the most.

My mother, the first pillar of strength in my life, who always believed in me and pursued me to do things that I love. My late father, who taught me a valuable life lesson on priorities. "*Be strong, be brave*" were his last words, and will stay with me forever, guiding me through difficult situations.

And most importantly, Nikita. You have, over time, become the prime pillar of my strength, and this work would not have been possible without you. Your motivation to keep pushing forward when things seemed tough has kept me going, and your support throughout this journey has been priceless. Thanks for standing by my side, and keeping me sane through this. I could not have done it without you.

Your whole life is full of things that you *could* have done, just not full of things you *have* done.

MR. BEAU BENNETT, THE TOUGH RANCHER, *The Ranch*

Whatever it is you seek, you have to put in the *time, the practice, the effort*. You must give up *a lot* to get it. It has to be *very important* to you. And once you have attained it, it is your power. It can't be given away: it resides in you. It is literally the result of your *discipline*.

MICHAEL CRICHTON, *Jurassic Park*

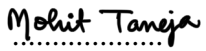
Life is *unpredictable*, not everything is in our control, but as long as you're with the *right* people, you can handle *anything*.

AMY SANTIAGO, *Brooklyn Nine-Nine*

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of Doctor of Philosophy, is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work.

Student ID: 20069777

Signed: 

Mohit Taneja

September 2020

Acknowledgment

We are formed as an average of the people we share our experiences with. I am lucky to have found a good support system both personally and professionally that have led me to be the person I am today.

First and foremost, my utmost thanks to my supervisor Dr. Alan Davy, who gave me the opportunity to pursue something that interested me. I am extremely grateful for his immense guidance and support throughout the ups and downs of this research journey. I owe it to him for giving me this opportunity to pursue a higher degree, for pushing me out of my comfort zone when needed, and keeping me inspired and motivated. Thanks for giving me the scientific intuition to think and analyze, and for providing all the facilities, support and critical skills necessary to be a researcher.

I am deeply grateful to Nikita for her inputs throughout the phase of this work. Without her support, dedicated efforts, and our innumerable brainstorming sessions, this would not have been possible.

I also extend my gratitude to Science Foundation Ireland (SFI), IBM and CISCO for funding my study, and Telecommunications Software and Systems Group (TSSG) for giving me the opportunity to learn from both academia and industry.

Last but not the least, I thank all my friends both in India and Ireland for their support, time and in/direct help in this journey.

Publications

• **[P1 - CF Procedia 2016]** M. Taneja and A. Davy, "Resource aware placement of data analytics platform in fog computing," *Procedia Computer Science*, vol. 97, pp. 153 – 156, 2016, 2nd International Conference on Cloud Forward: From Distributed to Complete Computing. [Online]. DOI: http://dx.doi.org/10.1016/j.procs.2016.08.295. Available: http://www.sciencedirect.com/science/article/pii/S1877050916321111

• **[P2 - IEEE/ACM SEC 2016]** M. Taneja and A. Davy, "Poster abstract: Resource aware placement of data stream analytics operators on fog infrastructure for internet of things applications," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2016, pp. 113–114. DOI: https://doi.org/10.1109/SEC.2016.44. URL: https://ieeexplore.ieee.org/document/7774693

• **[P3 - IM 2017]** M. Taneja and A. Davy, "Resource aware placement of iot application modules in fog-cloud computing paradigm," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 1222–1228. DOI: https://doi.org/10.23919/INM.2017.7987464. URL: https://ieeexplore.ieee.org/document/7987464

• **[P4 - WF-IoT 2018]** M. Taneja, J. Byabazaire, A. Davy, and C. Olariu, "Fog assisted application support for animal behaviour analysis and health monitoring in dairy farming," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Feb 2018, pp. 819–824. DOI: https://doi.org/10.1109/WF-IoT.2018.8355141. URL: https://ieeexplore.ieee.org/document/8355141

• **[P5 - CCNC 2019]** J. Byabazaire, C. Olariu, **M. Taneja**, and A. Davy, "Lameness detection as a service: Application of machine learning to an internet of cattle," in *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2019, pp. 1–6. DOI: https://doi.org/10.1109/CCNC.2019.8651681. URL: https://ieeexplore.ieee.org/document/8651681

• **[P6 - IEEE IoT Newsletter 2018]** M. Taneja, N. Jalodia, and A. Davy, "Towards Decomposed Data Analytics in Fog Enabled IoT Deployments - *IEEE Internet of Things*

Newsletter," September 2018, [Online]. Available: https://iot.ieee.org/newsletter/september-2018/towards-decomposed-data-analytics-in-fog-enabled-iot-deployments.html

• **[P7 - SPE 2019]** M. Taneja, N. Jalodia, J. Byabazaire, A. Davy, and C. Olariu, "Smartherd management: A microservices-based fog computing–assisted iot platform towards data-driven smart dairy farming," *Software: Practice and Experience*, vol. 49, no. 7, pp. 1055–1078, 2019. DOI: https://doi.org/10.1002/spe.2704. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2704

• **[P8 - IEEE Access 2019]** M. Taneja, N. Jalodia, and A. Davy, "Distributed decomposed data analytics in fog enabled iot deployments," *IEEE Access*, vol. 7, pp. 40 969–40 981, 2019. DOI: https://doi.org/10.1109/ACCESS.2019.2907808. URL: https://ieeexplore.ieee.org/document/8675283

• **[P9 - IEEE IoT Magazine 2019]** M. Taneja, N. Jalodia, P. Malone, J. Byabazaire, A. Davy, and C. Olariu. "Connected Cows: Utilizing Fog and Cloud Analytics Toward Data-Driven Decisions for Smart Dairy Farming," in *IEEE Internet Of Things Magazine* vol. 2, no. 4, pp. 32-37, December 2019. DOI: https://doi.org/10.1109/IOTM.0001.1900045. URL: https://ieeexplore.ieee.org/document/8982746

• **[P10 - COMPAG 2020]** M. Taneja, J. Byabazaire, N. Jalodia, A. Davy, C. Olariu, and P. Malone. "Machine Learning Based Fog Computing Assisted Data-Driven Approach For Early Lameness Detection in Dairy Cattle," *Computers and Electronics in Agriculture*, vol. 171, p.105286, 2020.DOI: https://doi.org/10.1016/j.compag.2020.105286. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016816991931840X

Fog Computing Support for Internet of Things Applications

Mohit Taneja

Abstract

With more devices on-board the Internet every day, there is a constant drive to balance Quality of Service (QoS) with an efficient use of resources. At present, the Internet of Things (IoT) applications are entirely hosted in the cloud. With emerging 'smart' scenarios in verticals such as dairy farming, health, home, mobility, etc., the real-time communication delay from the cloud platform necessitates the need to use computing platforms closer to the data source. While a traditional centralized cloud approach has led the path towards a pivotal revolution in modern-day computing, the emerging IoT era gave way to its own range of applications demanding a lower response time, efficient network usage, and improved data protection, to name a few.

In this age of IoT, the devices along the things-to-cloud continuum present a unique opportunity to additionally serve as computing hubs. Termed fog computing, this paradigm can be used to host applications and process data closer to the source. However, these intermediate devices are usually resource constrained in nature, and are thus limited in computational flexibility. This paradigm shift towards fog computing brings up a challenge of using these intermediary computing resources efficiently to host application(s) and serve as additional computational resources without affecting their primary functionality.

The research presented in this work addresses these demands and challenges, and presents how to use the fog computational platform to support these requirements. It presents a set of tools, algorithms, approaches and methodology of developing and deploying these emerging IoT applications while leveraging the fog computing paradigm. With extracting knowledge from the generated data being one of the prime objectives of IoT deployments, this work also presents how the data analytics computing operations can be decomposed to run on these resource-constrained devices without affecting their fundamental operation.

Contents

List of Figures

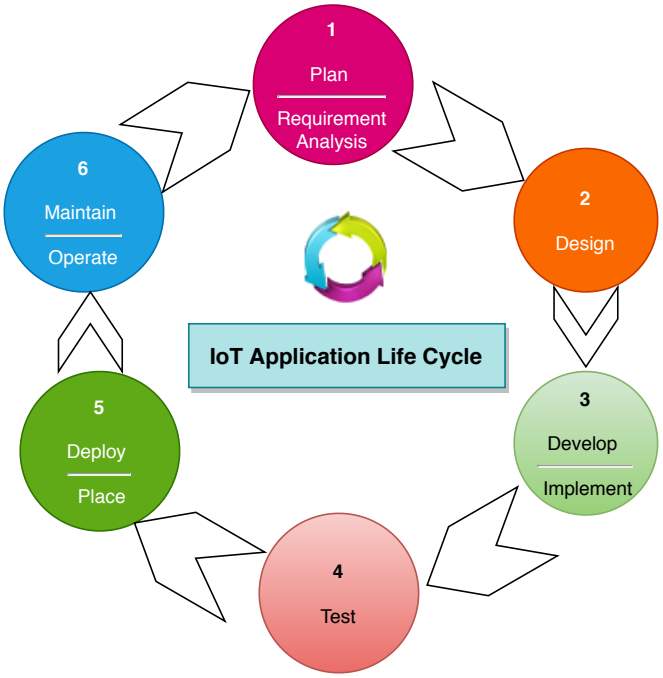
List of Tables

Chapter 1

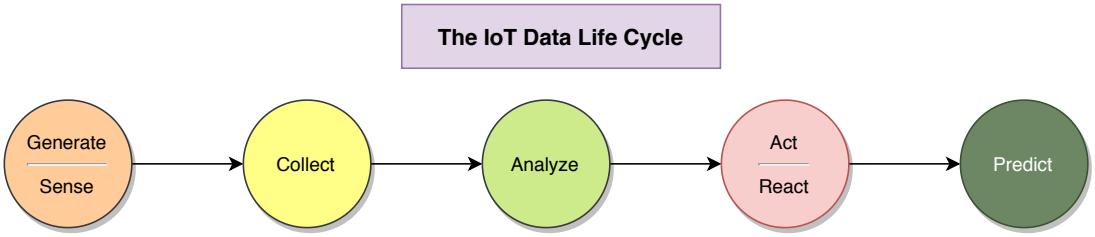
Introduction

With the exponential growth rate of technology, the future of all activities involves an omnipresence of widely connected devices, or as we better know it, the 'Internet of Things (IoT)'. IoT is about connecting people, processes, data and things; and is changing the way we monitor and interact. With the ever evolving IoT scenario, computing has spread to the most minuscule everyday activities, and is leading the way towards an everlasting future of connectivity and reachability. Moreover, the associated IoT applications are becoming an integral part of our day-to-day activities. IoT applications have been developed and deployed in several domains such as agriculture, transportation, manufacturing, logistics, healthcare, supply chain, factories and environmental monitoring to name a few. Despite their popularity and much progress, designing and developing IoT applications is still a complex, time-consuming and challenging task.

A generic IoT application life cycle inspired from a systems perspective has been presented in Fig. 1.1 [1], [2]. The systems development life cycle is a systematic process for planning, creating, testing, and deploying an information system. In the software engineering field, this is referred to as Software Development Life Cycle (SDLC) [4] or Application Development Life Cycle (ADLC), and in a broader perspective as Application Life Cycle Management (ALM). In the software engineering domain, SDLC is used as a systematic process for building software to ensure the quality and correctness of the software being built. ALM is a broader perspective than the SDLC; while the latter is usually limited to the phases of software development such as requirements, design, coding, testing, configuration and project management, ALM usually continues after development until the application is no longer used, and may span many SDLCs. To this end, a generic IoT application life cycle inspired from these concepts can be presented as shown in Fig. 1.1. A brief description of each of these stages of the IoT application life cycle is given below:



(a) IoT Application Life Cycle



(b) IoT Data Life Cycle

Fig. 1.1 A generic representation of a) IoT application life cycle (built and modified from [1], [2]), and b) IoT data life cycle [3].

1) Plan (Requirement Analysis): This stage deals with gathering the requirements of the system being developed. It includes inputs from all the stakeholders, domain experts, end-users that will either be part of the system being developed or will use it at a certain point in time.

2) Design: This stage deals with the architecture design, functional logic, inputs and outputs, database design, and addressing all types of dependencies for the system being developed.

3) Develop (Implement): After the design stage, the implementation stage deals with building the system using chosen programming language(s). Hence, this stage is a translation of the design document into source code.

4) Test: Once the development is finished, the system is tested for its functionalities. This is done to ensure that the entire system works according to the desired requirements specified in the earlier stage.

5) Deploy (Placement): After the testing, the developed system is deployed in the real-word to be used by the stakeholders.

6) Operate (Maintain): Once the system is deployed, and is in use, it needs to be maintained to ensure smooth operation as per the defined requirements. This stage may include – 1.) Re-configuration of the system, 2.) Bug fixing – any faults or issues that may arise, 3.) Upgrading the system to a newer version , 4.) Enhancement – to add some new features.

In the same Fig. 1.1, we also present the life cycle of data in an IoT deployment, sometimes also referred to as the IoT analytics life cycle [3]. Over here, the steps involve generation of data, followed by collection and analysis of the data, and the required action to be taken based on the analysis and the objective. The end goal of this analytics cycle is to have sufficient analytics ability that can make the predictions for the underlying use-case at hand. Extracting useful information from the data being collected from the IoT deployments is one of the prime objectives in many use-cases, thus it is important to understand the underlying life cycle of the process. More importantly, an IoT application might need to have a component that performs the desired analytics operation, and thus the understanding of data life cycle becomes useful in the design phase of the application life cycle.

Currently the IoT applications and associated services are delivered to consumers from the 'cloud', over the Internet and to our devices. The current cloud-centric approach alone cannot support a large number of current and emerging IoT applications, mainly for three main reasons outlined below:

1. First, the huge amounts of data produced by IoT deployments make it impractical to transfer it from the source (IoT end-devices) to the processing site (cloud servers) due to the underlying bandwidth limitation, processing overhead, and transmission costs.

2. Second, the significant end-to-end delay from end-devices to the remote cloud servers can hinder the performance of applications that require real-time analysis, such as online gaming, video applications, emergency response systems etc.

3. Finally, there might be implications over data transfer and handling in terms of privacy and security, and thus it might be desirable to avoid sending it remotely over the Internet; and in certain scenarios it might be forbidden as well.

To cope with these issues, fog computing paradigm has emerged as a new conceptual approach that combines the benefits of cloud, and the computational resources available in the infrastructure along the things-to-cloud continuum to provide services at the edge of the network.

As more and more devices with computing capabilities become connected to the Internet, these devices can also potentially participate in the hosting and delivery of these applications and services. In essence, the computing resources are moving beyond the 'cloud' and into the 'fog'. This new paradigm shift can bring many advantages if used effectively, such as lower cost, faster service delivery and improved data protection to name a few. However this paradigm shift requires a fundamental rethink as to how the applications are deployed and operated, and comes with its own set of challenges. This demands a methodology to be developed to use this paradigm shift effectively. To use these resources effectively and efficiently, there is need to have a careful placement of desired computing operations onto these resources, and careful coordination of them for efficient overall system performance.

It is important to understand both – the IoT application life cycle and the IoT data life cycle presented above in order to be able to leverage the fog computing paradigm over the continuum from IoT to cloud. In this work, we are mainly concerned with the first five stages of the IoT application life cycle, which can be defined at a higher level as **Design**, **Development** and **Deployment**. Further, from the IoT data life cycle, we primarily deal with the **Analytics** stage, and show how the fog computing paradigm can be leveraged for data analytics operation in IoT deployments. Fig. 1.2 shows a diagrammatic representation of these three subject-fields addressed in this dissertation.

We identify three key challenges to efficiently leverage the fog computing paradigm in IoT deployments as part of this work and address them in this research. The next section presents a detailed description of the articulated research questions and identified challenges. The work presented in the dissertation draws from three main subject areas — Fog Computing, Internet of Things (IoT), and Data Analytics. A collective diagrammatic representation of the work presented in the dissertation mapped to the subject areas is illustrated in Fig. 1.3.

The remainder of this chapter presents the research hypothesis that summates the intent of this dissertation, explicitly defines the scope using research questions, states the research contributions and outlines the organization of this dissertation.

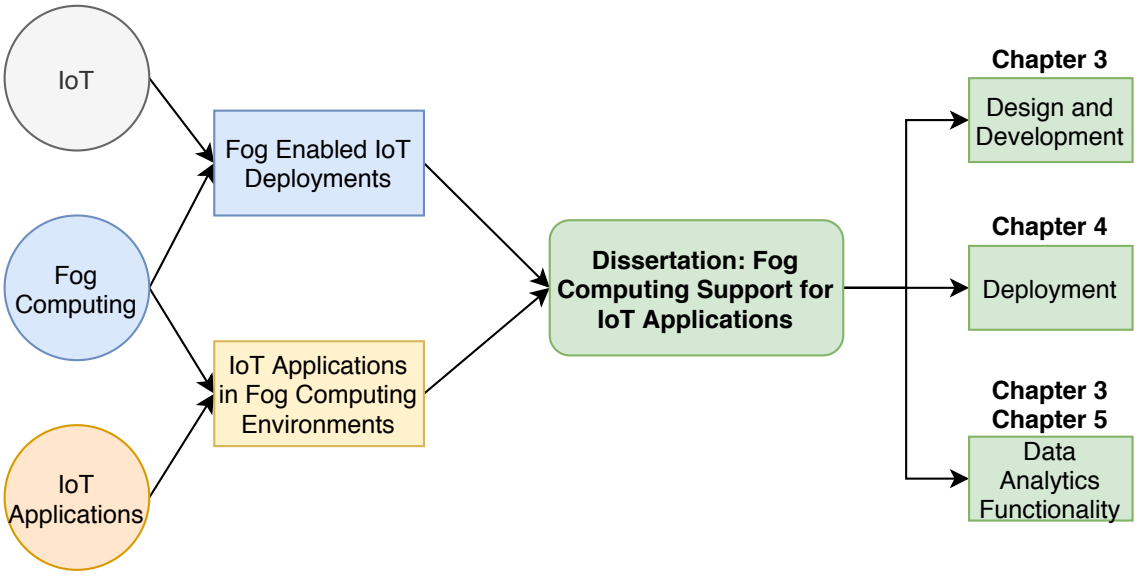


Fig. 1.2 A modular representation of the three subject-fields of the dissertation work. This represents the scope of the dissertation — IoT applications in fog computing environments and their: a) design and development, b) deployment, c) data analytics functionality.

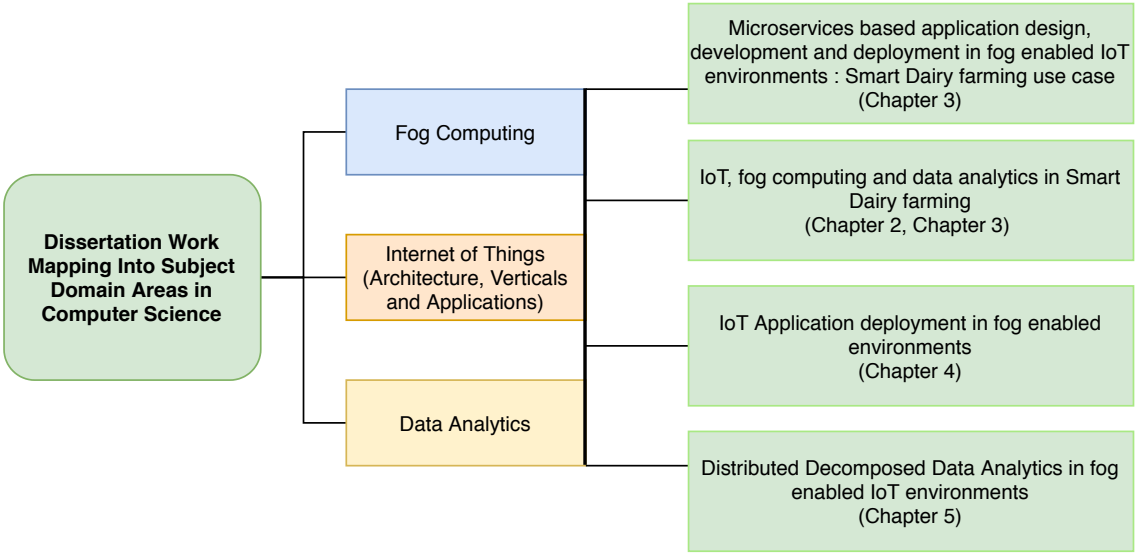


Fig. 1.3 A brief diagrammatic representation of this dissertation work mapped to key subject areas, and work presented in the further chapters.

1.1 Research Hypothesis

As mentioned earlier, the daily influx of devices getting connected to the Internet presents an opportunity to use them to perform various computing operations including data analytics. These devices if used efficiently can an bring many advantages, which builds up the research hypothesis of this work as below :

Executing parts of computing operations on fog nodes can reduce bandwidth consumption, cloud computing related resource and operational costs, and can help improve Quality of Service (QoS) for IoT applications.

Building up from the above, three research questions have been examined in this dissertation with their description as below:

1.1.1 Research Question (RQ1) — Designing and developing IoT applications leveraging the fog computing paradigm

One of the key challenges is to examine the development of IoT applications from the perspective of fog computing paradigm, where computing infrastructure available at the network edge can be used to host applications, or their parts. The key points of study here are to ascertain what methodologies or programming models to use while developing applications in such fog enabled IoT deployments. Current approaches being cloud-centric in nature do not fully serve the need here. The articulated research question thus is:

How to design and develop an IoT application leveraging the fog computing paradigm?

Give the wide scope of possible IoT applications, we selected Dairy Farming as the application domain to validate the approach. It was essential to decide on an application domain to have a realistic and time-bound approach for the work carried out. We validated the proposed approach in a real-world setup in a Smart Dairy Farm, but it is generic enough to cater to other application domains as well with the essential modifications depending on the underlying use-case scenario. Specific to the smart dairy farming application domain, the question can be re-stated as: *How to design and develop a fog enabled software system in a smart dairy farming IoT scenario, addressing a specified application objective?* The selected use-case and application objective was early lameness detection in dairy cattle using locomotion data. This has been elaborated in detail in chapter 3.

With the existing development methodologies being cloud-centric in nature, it becomes essential to investigate approaches that can be used and extended for fog enabled application development.

1.1.2 Research Question (RQ2) — Efficient deployment of applications on the computing resources available in the infrastructure leveraging the fog computing paradigm

Another challenge is how to deploy these applications efficiently onto available computing resources. The current approach of deploying them fully in the cloud does not consider the computing resources available in the infrastructure along the things-to-cloud spectrum. It does not take into consideration the resource capacity aspect of these available devices along the path for the deployment purpose. The primary reason for this is that the cloud has a resource-rich computing environment, where resources can be scaled up and down during run time. While on the other hand, these intermediate devices in consideration (termed as 'fog devices') to serve as additional computing hubs do not have the ability to scale resources on the go. Hence, a careful placement of computing operations is required for an efficient overall system performance.

How to efficiently deploy an IoT application in fog enabled infrastructure?

In fog enabled scenarios, the IoT application consists of various modules with active inter-dependencies that can be placed onto the computing resources available in the infrastructure. So, to make the research question more precise with the fog environment in consideration, it can be phrased as — *How to efficiently place components of a multi-component IoT application in the network infrastructure leveraging the fog computing paradigm?*

Using devices along things to cloud spectrum for hosting application components is not straightforward. The primary reason for that is the resource constraints with these devices, and not having the flexibility of the cloud to scale up and down resources dynamically for efficient resource utilization. Thus, there is a need to develop an approach that carefully places the application components onto available resources, while at the same time fulfilling their resource demand in order to provide the desired quality of service.

1.1.3 Research Question (RQ3)— Data analytics in IoT applications

In this IoT era, many "smart environments" (e.g., smart cities, smart factories) will be, among others, generators of huge amounts of data. To provide valuable services in such environments, data will have to be analysed to extract knowledge. Currently, this is typically achieved through centralized cloud-based data analytics services. However, this approach may present significant issues from the standpoint of data ownership, data protection and the network capacity. One possible solution to cope with these shortcomings is to move data analytics close to where data is generated i.e., on nodes that generate the

data or on nodes close by. Thus, it becomes essential to look into approaches that can help achieve this. Specific to the approach of decomposing data analytics for fog enabled systems that we present in chapter 5, the partial data analytics, termed as decomposed data analytics is performed at the edge of the network i.e., on a subset of nodes (which act as gateways and are termed as fog nodes) very close to the data source.

Extracting knowledge from data being one of the prime objectives of smart systems, thus raises the question of *whether the traditional data analytics algorithms can be distributed in some form onto the devices at the network edge to form an intelligent edge, capable of performing those analytic operations with the same computing principles as the traditional algorithm under analysis*. More formally,

How to decompose data analytics computing programs to run between fog and cloud?

The objective here is to perform efficient data analytics in IoT deployments, while leveraging the fog computing paradigm. Fog computing requires novel decomposition methods for computing programs. This decomposition between the cloud and fog is not straightforward, particularly for complex algorithms in data analytics applications. The unique system challenges such as storage, computation, communication are not considered by designers of machine learning algorithms, which makes the area worthy of investigation with fog computing in picture.

We sequentially address these identified challenges in the work presented.

1.2 Research Contribution

This section extracts and summarises the high level contributions that have been made as an output of this research work. The main contribution(s) of this dissertation are :

1. A design and development methodology for IoT application development leveraging fog computing have been presented. The proposed methodology has been validated in a real-world smart dairy farming IoT scenario.

2. With smart dairy farming as the selected application domain, contributions have also been made in behavioral analysis study of dairy cows. Specifically, a blended clustering and classification model for identifying lame cows have been proposed, with the proposed clustering methodology being major contribution as part of the research work from this dissertation. Although the existence of clusters in herd have been used before in cattle behaviours, the study presented here is the first to combine

that with a classification mechanism to build a custom machine learning model to detect lameness in dairy cattle at an early stage.

3. A framework for efficient placement of IoT application(s) in network infrastructure leveraging fog computing has been presented. The existing approaches deploy an application entirely in cloud even if the application is designed in modular way, composed of various modules. However, the novel approach developed as a part of this work presented in chapter 4 follows a hybrid fog-cloud strategy to efficiently use the computation platform available at the network edge for application deployment leveraging fog computing paradigm.

4. With realization of IoT deployments, and more and more data emerging as an output of that, data analytics becomes an integral part of the application being developed in these IoT deployments. An approach on how to enable these data analytics computing programs to run on devices available at network edge has been presented.

1.3 Dissertation Organization

The rest of the dissertation is structured as follows. Chapter 2 first presents related background information relevant for understanding this dissertation, followed by a literature review with respect to the areas of contributions.

The following three chapters address the research question in detail. Table 1.1 maps the research questions with the contributions made, and also indicates the corresponding chapter they have been presented in. Fig. 1.4 presents a visual representation of the same, along with the problem domain for each research question and the subject-field it contributes to.

Table 1.1 Research Contributions.

Research Question	Research Contributions	Chapter	Peer-reviewed Publications
RQ1	A hybrid modular-microservices approach for design and development of IoT applications leveraging the fog computing paradigm	3	P4 - WF-IoT 2018 [5] P5 - CCNC 2019 [6] P7 - SPE2019 [7] P9 - IEEE IoT Magazine 2019 [8] P10 - COMPAG 2020 [9]
RQ2	An algorithm for efficient placement of multi-component IoT applications leveraging the fog computing paradigm	4	P1 - CF Procedia2016 [10] P2 - IEEE/ACM SEC 2016 [11] P3 - IM 2017 [12]
RQ3	Extension of an existing approach for distributed decomposed data analytics in fog environments	5	P6 - IEEE IoT Newsletter 2018 [13] P8 - IEEE Access 2019 [14]

Chapter 3 addresses the first research question (RQ1). It presents the design and development of an IoT application in fog computing environments, and uses smart dairy

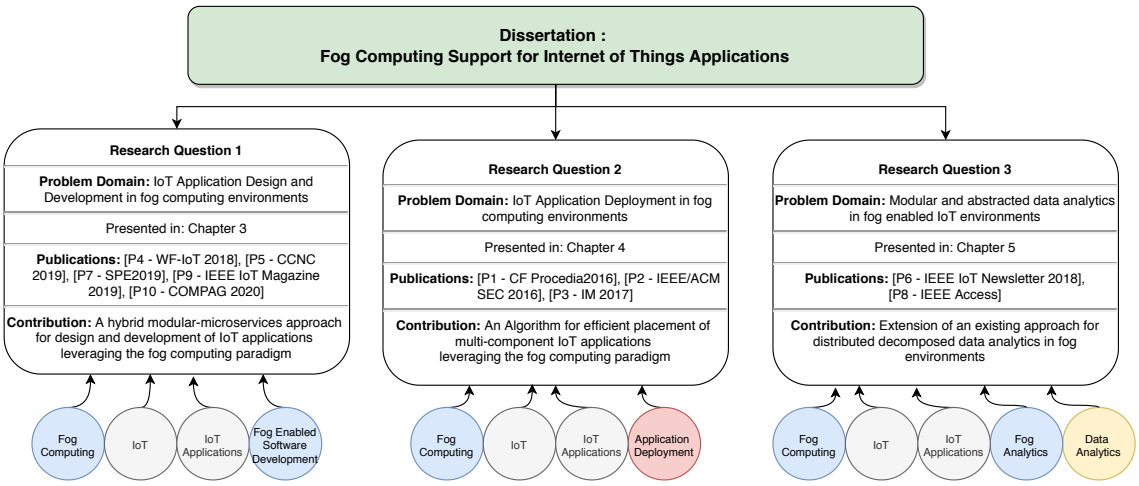


Fig. 1.4 A diagrammatic representation of dissertation work categorized into research questions, problem domain associated with the research question, scientific contributions made and also the peer-reviewed publications in which work presented in the dissertation was disseminated. The circles in the diagram represent the subject-fields it connects with.

farm as the IoT deployment and the application domain for the work. It introduces the key design principles of the proposed development approach and gives a complete methodical analysis done for the validation of the same. It also presents the underlying machine learning methodology used for the specified objective (early detection of lameness in dairy cattle) in the selected application domain of smart dairy farming.

Chapter 4 addresses the second research question (RQ2). It presents an efficient framework for deployment of IoT applications in fog computing environments. It describes the design, prototype, methodology and approach used in the development of the proposed framework. It puts forward an improved decision-making process of deployment stage of the life-cycle management of IoT applications.

Chapter 5 addresses the third research question (RQ3). It introduces how the data analytics functionality of IoT applications can further be modularized by means of decomposition of data analytics algorithms in fog computing environments. It presents the proposed distributed decomposed data analytics and its methodical validation in an experimental fog computing test-bed setup. It ascertains that if a computing resource can not host a complete data analytics service, then how it can be decomposed into smaller units in order to be able to run on resource-constrained devices in fog computing environments.

Finally, Chapter 6 concludes this dissertation by summarizing the chapters and presenting the limitations of the work, and directions of future work.

Chapter 2

Background and Literature Review

In this chapter, we present background, motivation, and literature review for the research presented in this dissertation. While the state-of-art is continuously evolving, this study helps in identifying knowledge gaps that demand further investigation. This chapter is structured as follows: section 2.1 to section 2.6 present the relevant background information, terminologies that are required to understand this dissertation, and also presents the motivation behind the research work. It presents an in-depth study of the concepts related to the main fields of research presented in this work. Further, section 2.7 presents an in-depth literature review in the contribution areas of this research in a structured manner in four sub-sections (2.7.1 to 2.7.4).

2.1 Internet of Things

The Internet of Things (IoT) is a network of connected physical devices, digital machines, sensors mounted on objects, wearables for animals or people that have the ability to transfer and exchange data with each other [15], [16]. It can be seen as a network of interconnected physical or virtual 'things' that are capable of using intelligent interfaces to be integrated as an information network in order to communicate with one another, with other devices and with services over the Internet to accomplish some objective [17]. Authors in [18] identified and envisioned the introduction and realization of IoT to affect both domestic and industry fronts. Some examples of domestic impact include application scenarios in assisted living, e-health, etc. On the industrial front, the areas of most visible impact include the likes of automation, industrial manufacturing, logistics, and intelligent transportation for people and goods.

In brief, a collection of 'things', capable of transmitting data to the nodes higher in the structural hierarchy of the network architecture can be defined as IoT.

A thing in the'IoT'can be a physical device such as temperature sensor, a person with body wearable or implanted device such as heart monitor implant, a farm animal with a wearable device, an automobile that has built-in sensors or any other man-made or natural object that can be assigned a unique identifier such as an IP address and provided with the ability to transfer data over a network. The simple idea is that anything that can be connected will be connected. It is noteworthy that there are other terms such as "objects", "Inter Connected Objects (ICOs)" [19] that have the same meaning as 'things' here, and are frequently used in IoT and fog related documentation. Some other terms used by the research community are "smart objects", "devices", and "nodes" [20].

IoT aims to bring every object online, hence generating massive amounts of data that can overwhelm the cloud centric application systems. In its report, McKinsey [21] estimates that the user base will have 1 trillion interconnected IoT devices by 2025, further substantiating the impending scenario. According to this estimate, by 2025 IoT will have a potential economic impact of USD 11 trillion per year, which nearly represents 11 percent of the world economy. As per a recent publication by Cisco [22], we have already reached the Zettabyte Era, and the number of devices connected to the Internet is growing exponentially. Thus, in near future, it has been estimated that a large number of applications will be processed and served through the realization of IoT [23].

2.2 Internet of Things Architecture

There is no universally-accepted overall architecture for IoT, although there are several efforts underway to achieve some convergence. IEEE P2413 [25], [26] (IEEE standards

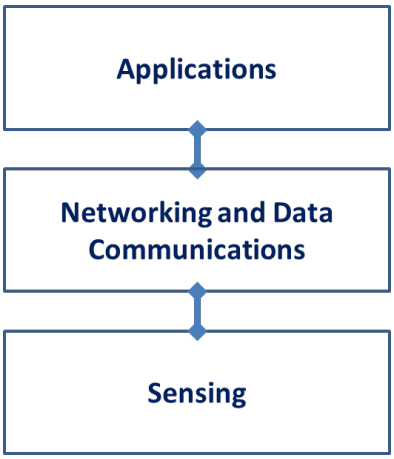
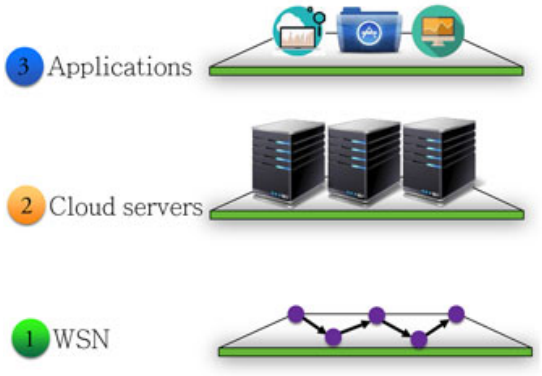
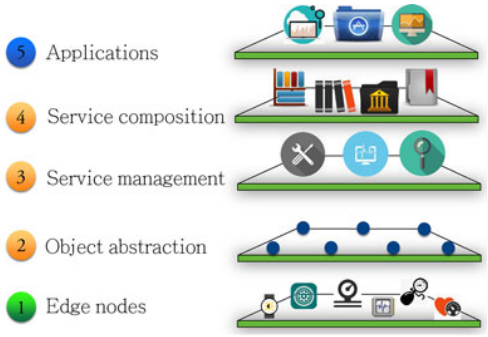


Fig. 2.1 Three Tier IoT Architecture by IEEE P2413 [24].

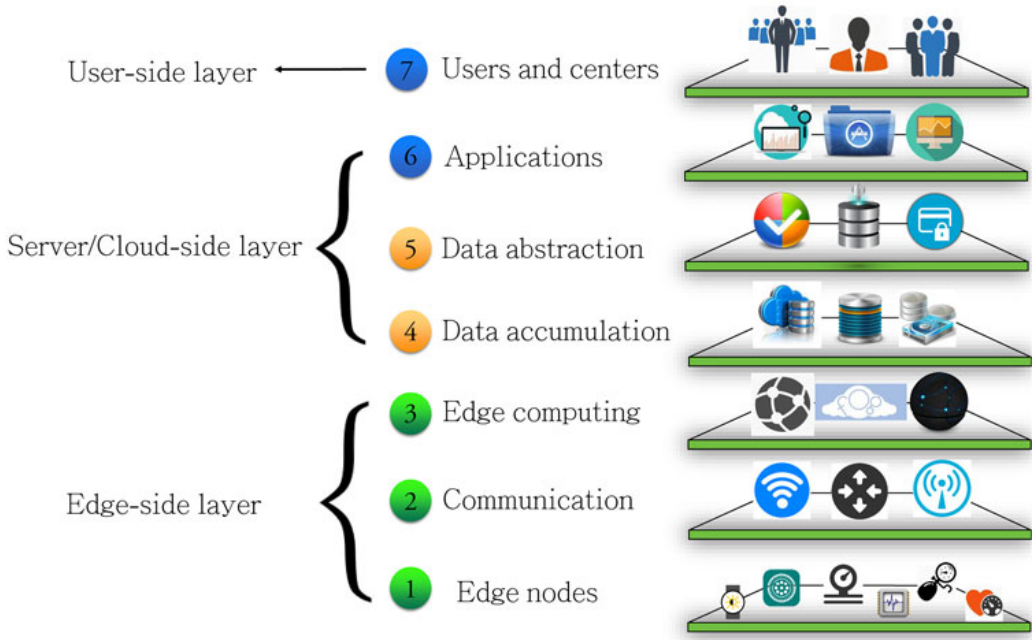
working group on IoT and IoT oriented applications design) considers the architecture of IoT as three-tiered [24], [27], with the layers as mentioned in Fig. 2.1.



(a) Three-level model [18]



(b) Five-level or Service Oriented Architecture (SOA) based model [28]



(c) CISCO's seven-level model [29]

Fig. 2.2 Three-level, five-level and CISCO's seven-level Internet of Things architecture reference models [30].

Several other IoT architecture models have been proposed in recent years as shown in Fig. 2.2 (adapted from [30]), and each of them focuses on some specific formulation or abstractions of the IoT ecosystem [31]. In 2014, CISCO presented a comprehensive extension to the traditional three, and Service Oriented Architecture (SOA) based five-level model by introducing their seven-level model [29] for the IoT paradigm. We consider the CISCO reference model as presented in Fig. 2.2c and Fig. 2.3 (adapted from [29]) in this study since it particularly summarizes the up-to-date modeling approach for IoT, and also covers the three-tier architecture of IoT as presented by IEEE P2413.

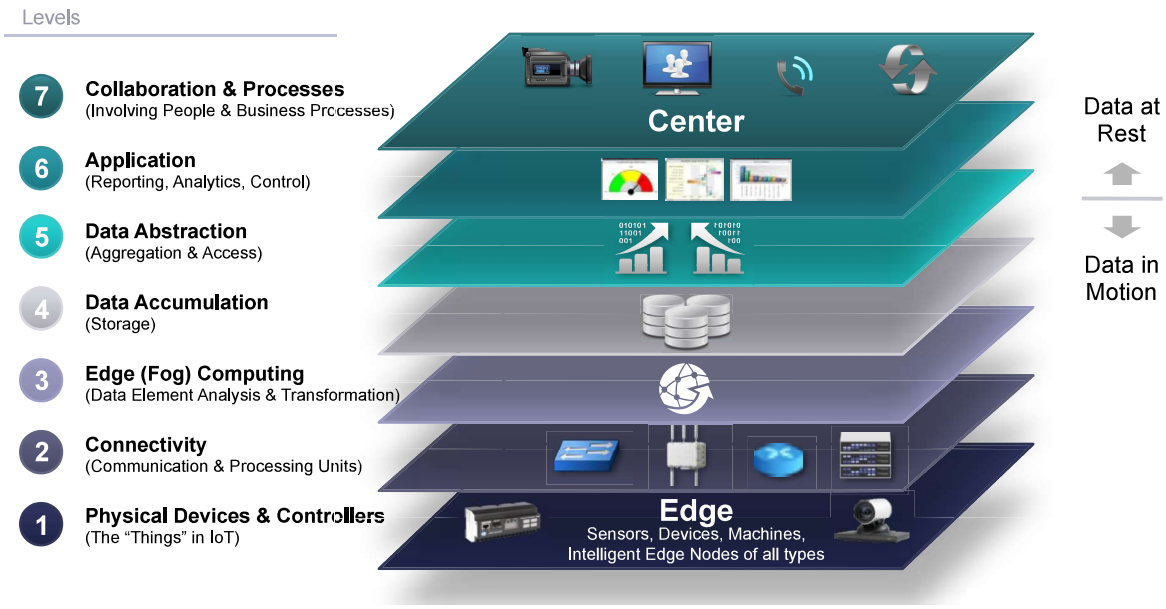


Fig. 2.3 The Cisco IoT Reference Model [29].

In all the architecture models, data flow is bidirectional in nature; however, the dominant direction of the flow of data depends on the application being worked on. For instance, in a control system, data and commands travel from the top of the model (the applications level) to the bottom (edge-node level), whereas in a monitoring scenario, the flow is vice versa (from bottom to top).

2.3 Limitations of Cloud-Centric Approach

Cloud computing has been a pivotal revolution in the field of computer science and research, revolutionizing the way software and applications work, and exponentially increasing the capability of the Internet in this related paradigm of hardware and resources. It has seen significant growth over the past decade, generating a shift from distributed network paradigm to being centralized, with its core being the data centres of cloud service providers [32], [33]. In the last few years, researchers [34] [35, 36] have illustrated the particulars of underlying processes involved behind the provisioning of cloud services.

Generally, the complete process of provisioning of cloud services involves invocation of several components (such as different Virtual Machines) within a data centre, and sometimes across multiple data centres dispersed in different geographical locations. Cloud based computing systems, applications and services are centralized in nature, and for every request, service provisioning may involve one or more data centres. With data centres as the hub of computing resources and data centre networks invoked every time an application makes a service request, it is inevitable for such a system to disappoint with the

increase in IoT user-base demanding real-time responses. The centralized cloud-centric outlook for emerging real-life IoT scenarios thus seems infeasible and unprogressive with the realization of IoT.

The increasing range of real-world IoT deployments essentially increases the sources of data generation, thereby globally strengthening the challenges already being faced in the big data space [37], particularly regarding moving data from one end (i.e., from data sources such as sensor/IoT devices at the edge level of infrastructure) to the other extreme end (i.e., centralized data centres at the cloud) in the network infrastructure. Therefore, building architectures that allow executing services in multiple points have recently gained attention from industry players [38], [39].

Sending the entire data set across the extreme ends in the infrastructure becomes an unrealistic solution, specifically in scenarios with constrained network bandwidth and scarce Internet connectivity. Instead, approaches that collect data and perform computational processing near the data source itself present a more practical alternative in such scenarios. This generates a vast array of benefits across use cases such as those with video oriented applications, where the transport of video across infrastructure can claim considerable network resources such as requirement for storage at each node from source to destination.

Thus, despite its advantages, the rapid increase in ubiquitous mobile and sensing devices which are connected to the Internet challenges the traditional network architecture of cloud computing framework. With IoT in play, we will have billions of interconnected heterogeneous devices emitting large volumes of data streams for processing. In such a scenario, transferring all the raw data to cloud for analysis is neither scalable nor suitable, especially for real-time decision processing. Thus, the emerging IoT applications demanding dynamic scalability, efficient in-network processing, and real-time low latency communication have led to the evolution of the **fog computing** paradigm, which has been discussed in detail in the next section.

2.4 Fog Computing

With the contemporary trends in data volume, velocity, and variety and the limitations of cloud as mentioned above, Cisco proposed the revolutionary concept of Fog Computing [40]. Fog computing aims to provide traditionally centralized data-center operations at the edge of the network. It is a relatively new networking paradigm and has recently emerged as a potential architecture for scaling IoT network applications. A graphical representation of a general fog computing architecture with fog nodes forming a layer between IoT devices and the cloud has been illustrated in Fig. 2.4.

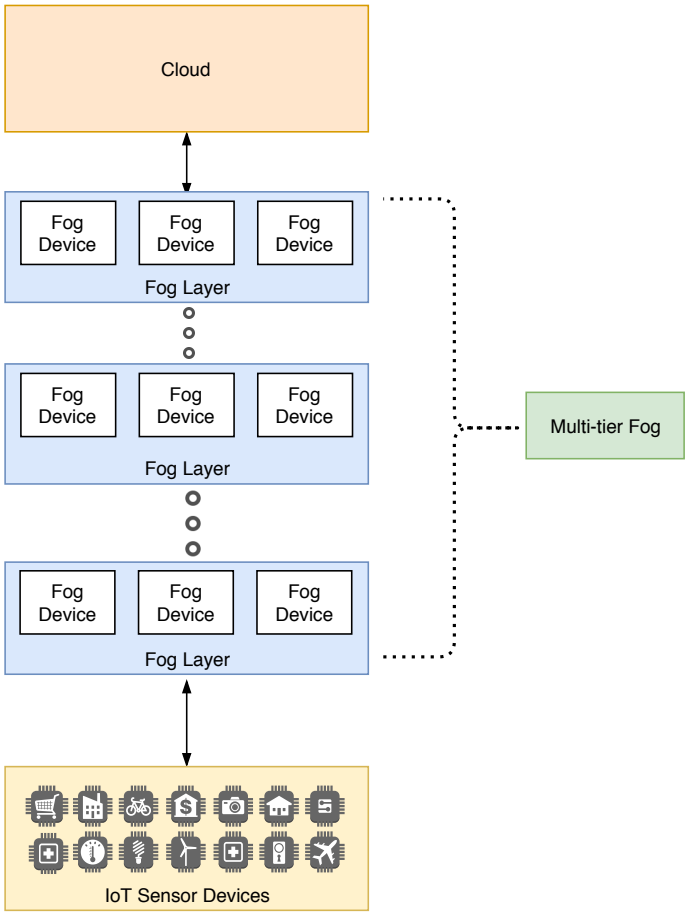


Fig. 2.4 Fog computing architecture representation as three tier IoT-fog-cloud architecture with multi-tier fog in a standardized IoT reference architecture model [41], [10], [12].

2.4.1 Fog Node

Several interpretations [42] have been proposed for the implementation of fog nodes and their configuration, either via servers [43, 44], networking devices [45–47], cloudlets [48, 49], base stations [50, 51], or vehicles [52, 53]. These interpretations exist because of IoT ecosystem being multidimensional and multidisciplinary in nature. Owing to rapid expansion and a vast expanse of possibilities, an IoT ecosystem is hard to define, complex, and difficult to capture [54]. To this end, a graphical representation of one of the possible multidimensional view of the IoT ecosystem illustrating IoT devices, available deployment platforms and type of IoT applications has been illustrated in Fig. 2.5.

To our understanding, a fog node is any computing device that can be used to deploy applications or a component of an application on it— for example, depending on the specific use-case, it could be a gateway, set-top-box, switch, router, PC, etc. So, in the network architecture and infrastructure, any element that is capable of performing any

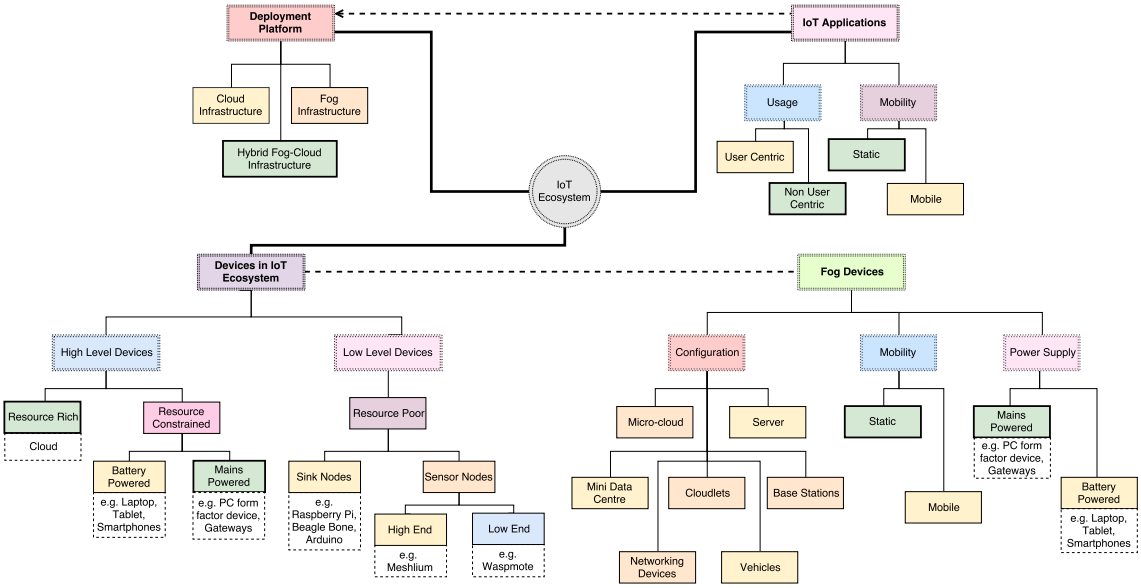


Fig. 2.5 Visual representation of IoT Ecosystem illustrating IoT devices, available deployment platforms and type of IoT applications [7].

computing operation such as being able to host and run an application module(s) is considered as a fog node or fog device.

Fog computing aims to distribute data, computational processing and application to the devices located at the network edge, which traditionally used to exist almost entirely in cloud. It utilizes the available in-network computing resources and shows the capability of reducing the dependency on the cloud by facilitating various computing operations on the network edge [40] depending on the use-case and deployment. According to the Open Fog Consortium [55] (now known as Industrial Internet Consortium [56]), the fog nodes are not completely fixed to the physical edge, but should be seen as a fluid system of connectivity [41]. A survey by the authors in [57] gives an overview of the opportunities and challenges of fog and IoT.

The capabilities of the computing gateways [58] comes as an important aspect of fog computing architecture for IoT based applications. With the increasing demand of providing smart solutions, the shift towards utilization of gateway devices as fog nodes for performing computing operations such as edge analytics is being supported by an increasing number of industrial [19], [59], [60] IoT platform developers and solution providers, including IBM, Intel, and Microsoft.

2.4.2 Fog Enabled IoT Deployments and Tier-Division in the Architecture

A fog enabled deployment means that there are elements in the network that are capable serving as fog node(s), as per the understanding of a fog node presented in previous section (2.4.1).

At an abstract level, fog computing architecture as presented in Fig. 2.4 consists of three tier network structure as below:

1. **Tier 1 - Lowest Tier - IoT Layer** consists of IoT devices generating data streams.
2. **Tier 2 - Middle Tier - Fog Layer** comprises of fog devices in fog layer.
3. **Tier 3 - Highest Tier - Cloud Layer** is the cloud (traditional data-centres).

A fog enabled deployment may consist of several tiers of nodes [41], leading to multi-tier levels in fog layer as presented in Fig. 2.4. The number of tiers in a deployment are use-case dependent. A detailed explanation about this has been given by OpenFog Consortium [55] in OpenFog Reference Architecture [41] released in 2017. The text and points below outline in brief how use cases determine number of tiers in fog deployments.

Fog deployment can be small scale or large scale based upon the given scenario being addressed. The number of tiers in a fog deployment will be dictated by the scenario requirements, including:

- Amount and type of work required by each tier
- Number of IoT devices
- Capabilities of the nodes at each tier
- Latency between nodes and latency between sensors and actuation
- Reliability/availability of nodes

In general, each level of the N-tier environment would be sifting and extracting meaningful data to create more intelligence at each level. Tiers are created in order to deal efficiently with the amount of data that needs to be processed and provide better operational and system intelligence.

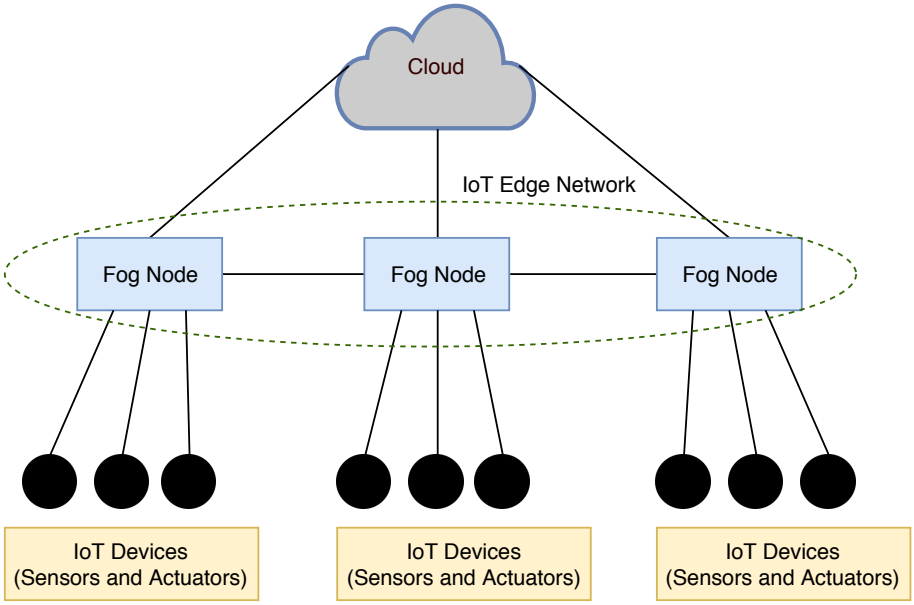


Fig. 2.6 Tree Topology in Fog Enabled IoT Deployments [14].

2.4.3 Tree Topology Representation in Fog Enabled IoT Deployments

Fog enabled IoT deployments can be visualized and modelled as a tree topology representation. Within scope of the work presented, we consider a tree like network architecture in which an IoT device (say i) is connected with its unique fog node (say j) which is further connected to the cloud. We have sensing devices that are transmitting their data to fog nodes and they are sending their data towards cloud or another central location. The tree topology is a hierarchy of nodes with single root node at the highest level of hierarchy, which is connected to one or many nodes in the level below. The communication, computing and storage capabilities in node(s) increases as one moves from branches of the tree towards the root node. In this tree-like topology, the root represents cloud, intermediate nodes represent fog nodes and the leaf nodes represent IoT devices. Data in IoT based deployments moves from ‘ things ’ to cloud, or in terms of tree representation, from branches of the tree to cloud via fog nodes, allowing data to be processed closer to where its generated. As presented by the authors in [61], the benefits of tree topology include the fact that it is scalable in nature, and has a simple structure that makes it easier to identify and isolate faults. The graphical representation of an end-to-end tree like network architecture is as shown in Fig. 2.6. It should be noted that there can be topologies inside each layer/level but the main structure and overall abstract topology is tree-like.

2.5 IoT Application Architecture in Fog Enabled Environments

There are two essential pieces to understand when it comes to developing a software or application addressing a particular scenario. First is ***System Architecture*** and second is ***Application Architecture***. This is irrespective of whether the solution being developed is dependent on data generated from IoT, or whether it is cloud based or fog based. The above plays an important role in all cases. The details on these have been presented below:

1. **System Architecture:** The system architecture concerns with the infrastructure in a deployment. It usually comprised of several entities and also highly depends on the use case and scenario being addressed. The main entities for an IoT based fog enabled scenario consists of *IoT devices, fog nodes and cloud data centers (cloud itself can have entities such as resource manager and virtual machines).*

2. **Application Architecture:** The application architecture concerns with design of the solution, programming approaches, and various entities from system architecture on which the application will be deployed and running. The application architecture is also dependent on the use case and specific scenario being addressed. While there can be some similarities between different application architectures, there is no universal one-size-fits-all architecture that is suitable for every scenario. Applications might be different in functionality but might have a common application architecture, and even system architecture.

   Both of these are highly dependent on the use-case and specific scenario being addressed.

We specifically focus on **IoT based applications**[1] here i.e., applications that are dependent on IoT devices as a data source.

Thus, to our understanding, an IoT application can be defined as an application that has an IoT element associated with it. More specifically, within the scope of the work, we consider IoT applications that operate on data as input generated by IoT devices (sensors, etc. in Tier-1 of the fog computing architecture presented in Fig. 2.4).

2.5.1 Distributed Multi-component IoT Application Architecture

Fog-enabled IoT applications are usually divided into multiple interconnected application modules [62]. Program execution can naturally be described as a graph in which vertices

[1] We use IoT based applications and IoT applications interchangeably in this document, both of them refer to the same thing.

represent computation that are labelled with computation costs and edges reflect the sequence of computation, sometimes also labelled with communication costs [63] where computation is carried out in different places.

Thus, in a similar way applications can be described as composed of set of independent parts (or can be divided into set of independent parts) performing computation and having active dependencies. The granularity of these independent parts may vary from application to application and is majorly dependent on application functionality, these independent parts can be a component or module [64], or Java class [65], or even a method [66].

To our understanding and within the scope of the dissertation, a ***distributed multi-component IoT application*** *can be defined as a set of software components running on different nodes (and also might be in different geographical locations) in system architecture (i.e., the infrastructure) and connected to each other through networking technologies.*

Some examples of applications made up of independently deployable computing units are QR-code recognition [67], health monitoring using body sensor network [68], mobile augmented reality application, etc. The number of independently deployable computing units vary from application to application, for e.g., in a mobile augmented reality application, the number of such components amounts 5 to 10 [69], with different components for tracking camera movements, building a 3D map of the environment, recognizing objects (object recognition module), detecting collision between objects (collision detection module), rendering a 3D overlay etc. Other applications such as 3D games are reported to consist of 10 to 20 components [70].

2.6 Communication in Fog Enabled IoT Environments

Even though our intention is not to survey different communication protocols, we briefly present major communication protocols in the Fig. 2.7 that are used in IoT based applications and, especially, in the fog computing domain. It is important to note that none of these protocols is superior to another. This is simply because all of them are superior in some aspects and weak in other aspects [19]. Ideally, each protocol is designed to support a particular use-case scenario more efficiently than others. Fig. 2.7 below illustrate how different protocols fit in with respect to Open Systems Interconnection (OSI) model. Some of the protocols are designed to communicate over short distances and are more suitable to conduct communication between things and fog nodes. Other protocols are typically suitable to communicate between fog nodes and the cloud infrastructure. As a result, in fog computing, fog nodes should be able to deal with different types of communication protocols, and mostly in parallel.

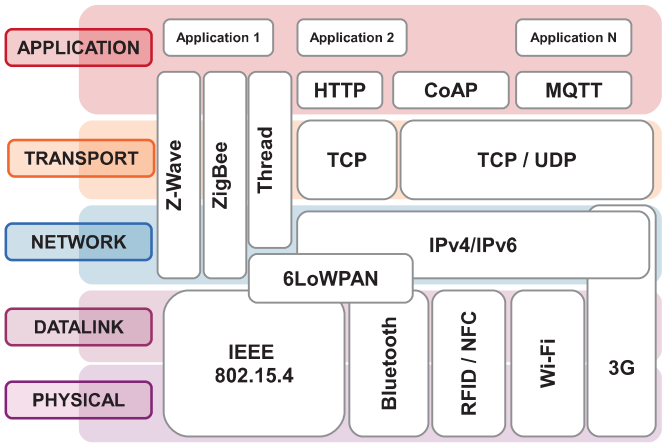


Fig. 2.7 Overview of Communication Protocols with respect to OSI model [71]

2.7 Literature Review in Contribution Areas of the Thesis

This section presents a literature review done for the main contribution areas of the dissertation. These form important aspects for the presented work and help lay a foundation for the contributions made through our research work. Fig. 2.8 gives a graphical representation of the literature review performed in the three contribution areas of the dissertation.

2.7.1 Design and Development of IoT Applications in Fog Computing Environments

The challenge of designing, developing and deploying software for the Internet of Things (IoT) is often underestimated [72]. For IoT systems to deliver their full potential, developers must leverage the computing resources available along the things-to-cloud spectrum in an IoT deployment. In the IoT vision, applications are no longer isolated, proprietary silos of devices and software. Instead, they must combine resources as features that are readily available in the environment: some generic, some application specific, and some legacy things [73].

2.7.1.1 Fog-enabled IoT Applications from Practical Standpoint - Microservices Based Approach and Modular Application Design Approach

As mentioned previously, in fog enabled environments an application can be considered as a collection of application modules or components [74], [42], [62]. Thus, it is also necessary to understand software development methodologies that can be used to develop such applications for implementation in real-world systems.

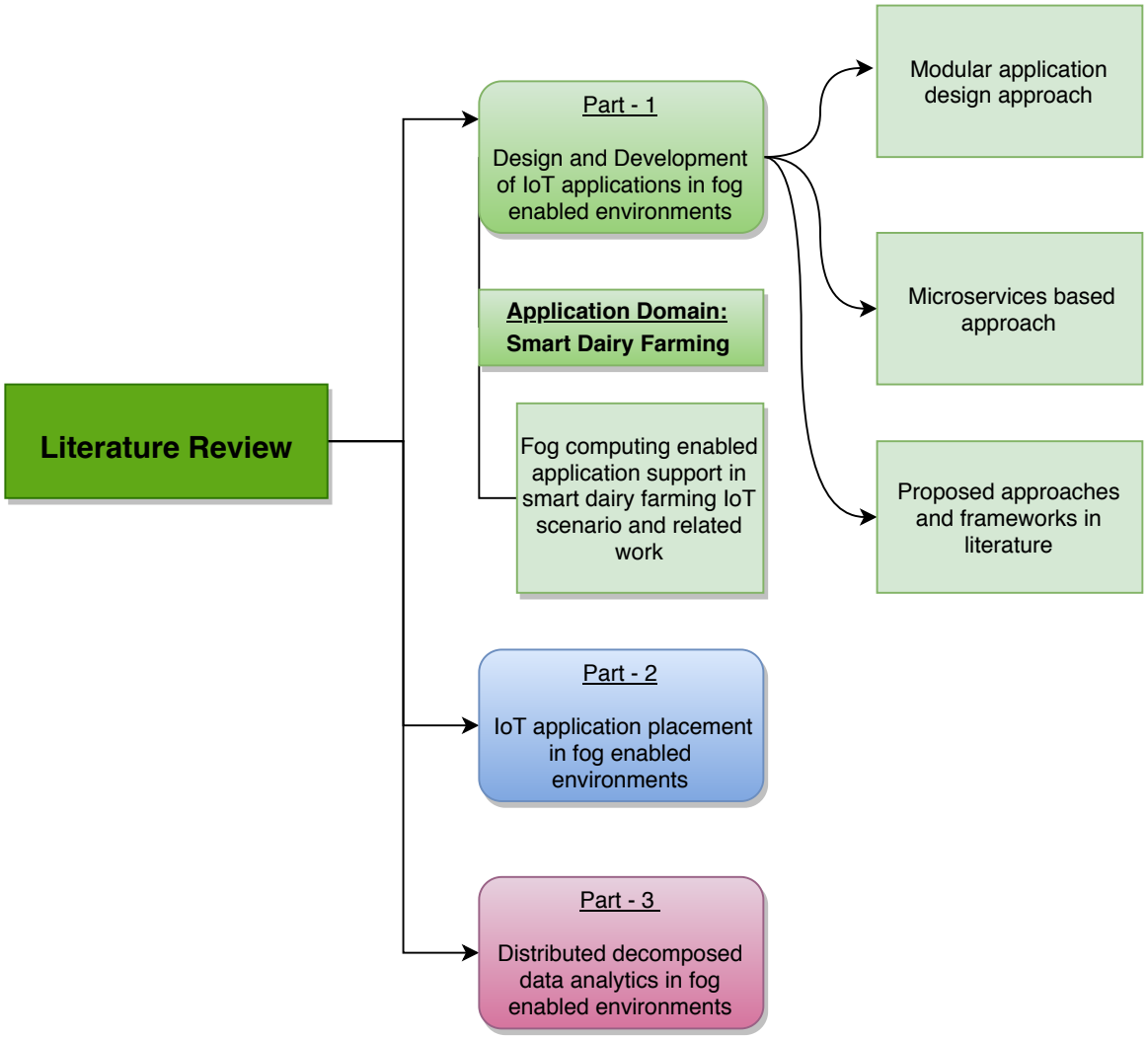


Fig. 2.8 A graphical representation of literature review performed in contribution areas of the dissertation.

Microservices Approach [75]: Microservices based approach focuses on building an application as a set of set of smaller, interconnected services. A service typically implements a set of distinct features or functionalities. Each microservice might have its own business logic along with various inter-dependencies. Some microservices would expose an API (Application Programming Interface) that is consumed by other microservices or directly by the user.

Modular Approach [76]: Modular application design approach subdivides a system into smaller parts, called modules, which can be independently created, modified, placed and replaced or exchanged between different systems. Each module is a separate software component. It emphasizes on breaking large problems or programs into smaller modules/units to increase the maintainability, and to make it easier to allow any changes like addition of new features to be made in future.

While microservices based approach implements the modular approach by its inherit nature of the move towards modularity, but it is ideal to list and understand them separately given that they have existed as two discrete items in the literature. The main aim here is to understand the fundamental concepts behind these two approaches, and develop an approach that can be used to design and develop IoT applications in fog computing environments.

To our understanding, microservices based approach is more practical from an implementation stand-point, while modular approach is more conceptual in nature, and is applicable in wider scenarios than just software development. The concept from the modular design approach can be combined with the microservices approach to build a more practical approach for IoT application design and development in fog computing environments. We believe that a microservices based approach would be adept at the devel-

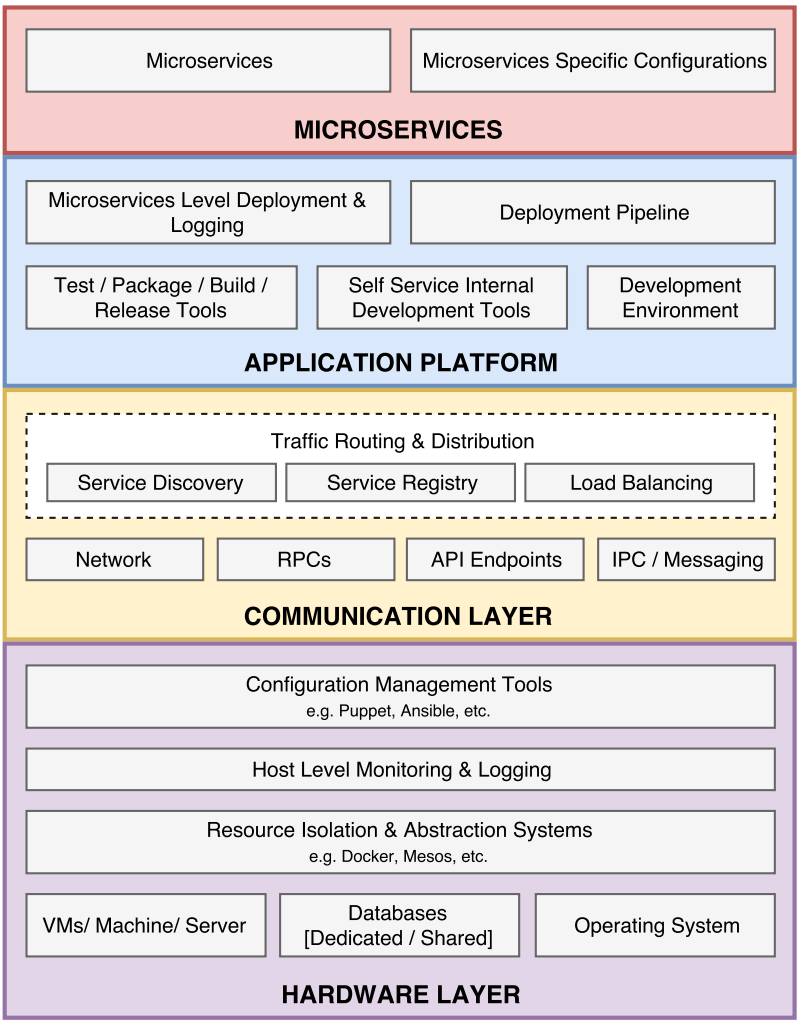


Fig. 2.9 The four-layer model of microservice architecture- A comprehensive breakdown of what lies in the four layered microservices architecture [75], [7]. API, application programming interface; IPC, interprocess communication; RPC, remote procedure call

opment and deployment of IoT applications over fog enabled infrastructures. This involves a proposed change in the conventional application design and development, whereby an IoT application can be built as a collection of microservices or decomposed into a collection of microservices for an already existing application, which can then be distributed across physical resources available in the cloud and on the network edge. Traditionally, all components of an application run in data centre (cloud). With fog computing into play, component of application can be deployed and run across physical hardware resources available in the cloud and on the network edge. Ever since the introduction and adaptation of DevOps in practice, the microservices architectural style is the first realization of a Service Oriented Architecture (SOA) in this context, and with the plethora of advantages that fall in line with the flexible future applications and service, is rapidly evolving to be the standard for developing continuously deployed systems. Some key reasoning outlining similarities in the objectives of microservices and IoT are as follows:

- Lightweight communication
- Independent deployment facility
- Bare minimum centralized management and control
- Isolation support
- Building one or multiple applications from a set of different services
- Technological independence and support for multi-vendor interoperability

The microservices architectural model can be depicted to be an abstraction of four [75] layers— hardware, communication, application platform, and microservices. Each of these layers comprises of components as shown in Fig. 2.9. One of the key takeaways of the microservices architecture is technological independence— each service is independent in terms of deployment and platform, and potentially the technological stack. They can run their own tasks and processes, and communicate via lightweight protocols and APIs. This enables each service to be a business capability that can potentially use its own technological stack, but still be a part of a connected comprehensive application structure towards a solution to a larger use case.

As we can see that the architectural goals of both microservices and IoT are quite similar. Although the practice instead sometimes can be different as can be seen from the comparison presented in Fig. 2.10 (taken from [77]).

Altogether, the microservice approach comes from a different direction than that of IoT, but both have the same architecture goal. A detailed consideration is required here in order to be able to incorporate this approach into IoT, especially in fog enabled IoT scenarios.

Feature	Microservices	IoT / CPS
self containment	wrap around business domain and reduce dependencies, libraries packed with application	often around device capabilities, libraries not packed with application
orchestration vs. choreography	choreography preferred	often orchestration (esp. when using HTTP, DPWS or CoAP)
container virtualization	yes, Docker, lxc, etc. for separation, scalability and ease of deployment	no, but similar, OSGi
continuous integration	yes, test overall application	partly, e.g. in single vendor scenarios
continuous delivery	yes, short release cycles	no, rare updates
protocols	HTTP	HTTP, MQTT, CoAP, DPWS

Fig. 2.10 Comparison of Microservices and IoT/Cyber Physical Systems (CPS) [77].

As mentioned that the microservices architectural style comes as the first realization of a service-oriented architecture and is currently in wide use by industry for software development and deployment as part of best DevOps practices. But its use has been limited to cloud based applications only, i.e., deploying all the developed microservices for an application within the data-centre itself, and very limited work [78], [79] has been done that focuses on the deployment of applications across the cloud and network edge, especially in the case of IoT applications. The cloud based applicability of a microservice style architecture to design a smart city IoT platform has been presented by the authors in [80], where they share their experience of implementing microservices approach in the cloud environment. But all such cases so far have only been limited to the cloud, and not explored in context to the fog computing environments.

The **challenge** here is how to apply the microservices approach to build the application in an IoT scenario leveraging the fog computing paradigm. That is where combining the concept of modular design approach with the microservices design principles comes into play, and helps in building a hybird modular-microservices based approach for IoT application design and development in fog computing environments. This is where we position a part of the work presented in Chapter 3 as a contribution in the field.

2.7.1.2 Proposed Approaches and Frameworks

A number of different application design, modelling and development approaches [81] have been proposed in the literature for IoT based applications, primarily in cloud computing environments and a few in fog computing environments. A brief description of those have been presented below:

- **PatRICIA [82] :** Authors [82] describe PatRICIA (PRogramming Intentbased Cloudscale IoT Applications) as a framework to define an ecosystem, which provides an end-to-end solution for cloudscale IoT applications. The core idea of the PatRICIA is to enable the development of value-added IoT applications, which are executed and provisioned on cloud platforms but leverage data from different sensor devices and enable timely propagation of decisions, crucial for business operation, to the edge of the infrastructure.

- **IDeA [83] :** IDeA [83] adopts a model-based systems engineering methodology for IoT application development, focusing on the design phase. It consists of a method called IoT DevProcess and a supporting tool called IoT AppFramework. It provides high-level abstractions to address the heterogeneity of hardware devices and software components in the system model through a SysML profile named SysML4IoT. SysML4IoT aids stakeholders to deal with the system complexity and unambiguously communicates the system model. Furthermore, it promotes reusability and interoperability amongst software components and systems. IDeA addresses the concerns of the various stakeholders (device expert, domain specialist, requirements engineer, application engineer and deployment manager) using Views and ViewPoints. The IoT application Viewpoints, SysML4IoT, and the model library make up the IoT AppFramework. The IoT DevProcess is an extension of the object-oriented system engineering method, which allocates activities to predefined stakeholders, provides modifications and inclusion of existing activities and artefacts.

- **FRASAD [84]: Framework for sensor application development :** FRASAD [84] is a node-centric software architecture and a rule based programming model that allows designers to describe IoT applications. It uses the multi-layered model driven architecture (MDA) for its architectural design. At the highest abstraction level, a domain-specific language (DSL) enabled by rule-based programming model uncouples the programming language and the execution model used by the underlying operating system (OS).

  FRASAD supports the interaction between an upper layer and the next lower layer through a predefined interface. Every layer is encapsulated into a component. IoT applications are built upon this architecture, independent of the underlying platform. A translator/compiler over the OS layer deals with the complexity of the translations between both abstraction levels. However, in evaluating FRASAD, there is a need to extend the programming model and the framework to more operating systems and different kinds of IoT applications.

- **DataTweet [85] :** DataTweet [85] is an IoT application development framework that provides a mechanism for decoupling an IoT Application Logic (AL) from common IoT functionalities such as discovery, configuration, service management and registration. This framework provides an open source API for the common functionalities, which allows developers to concentrate more on the end-user centric aspect of the IoT application. The API significantly reduces the amount of written code and ultimately the time to complete the application. The framework mainly consists of two components: (i) AL and (ii) common service entity (CSE). The AL functionalities are primarily implemented by RESTful web services. The CSE is responsible for the identification and combination of several common functionalities required in the development of an IoT application. The CSE interacts with the hardware devices and provides a standard API to developers. To promote cross-domain/use cases IoT application scenarios, this framework can allow the CSE of a particular domain to be connected to embedded devices belonging to a different domain of operations. Furthermore, the AL belonging to one application can be made to communicate with the CSE of another application, thus enabling horizontal IoT use cases. However, in industrial applications, this feature may be difficult to achieve.

- **Moblie Fog [74] :** Authors [74] describe Mobile Fog as a high level programming model for future Internet applications that are geospatially distributed, large-scale, and latency sensitive. Many large-scale future Internet applications require location and hierarchy-aware processing to handle the data streams from widely distributed edge devices. It provides a programming abstraction and allows applications to easily use fog resources while supporting dynamic scaling at run time. In Mobile Fog, an application consists of distributed Mobile Fog processes that are mapped onto distributed computing instances in the fog and cloud, as well as various edge devices. While running, each process performs application–specific tasks such as sensing and aggregation with respect to its location and level in the network hierarchy. The Mobile Fog communication API is composed of event handlers that must be implemented and standard functions that can be called by the application. The same code can be run on a different device like smartphones, vehicles, or cameras; the developer only needs to write the code once.

- **Foglets [86] :** Foglets [86] is a programming model that facilitates distributed programming across fog nodes. It provides APIs for spatio-temporal data abstraction for storing and retrieving application-generated data on the local nodes. Through the Foglets API, their processes are set for a certain geospatial region, and the application components are managed on the fog nodes. Foglets is implemented through

container-based virtualization. The Foglets API takes into account QoS and load balancing when migrating persistent (stateful) data between fog nodes.

- **Distributed Data Flow (DDF) [87], [88] :** Dataflow [89] is a well-known programming model that has been applied for developing Wireless Sensor Networks (WSN) applications in several works [90]. In the dataflow programming model, application logic is expressed as a directed graph (flow) where each node of the directed graph can have inputs, outputs and independent processing units. There are nodes that only produce outputs and ones that only consume inputs, which usually represent the start and the end of the flow. The nodes processing units process the inputs and produce outputs for downstream nodes. The processing unit of a node executes independently and does not affect the execution of other nodes. Thus, the nodes are highly reusable and portable.

  A DDF is a dataflow program where the directed graph (flow or nodes of this directed graph) is deployed on multiple physical devices rather than one. Each physical device may be responsible for the execution of one or more nodes in the graph, and may form sub-graphs. Some inter-node data transfer may happen between devices.

  For the IoT, the dataflow programming model offers a significant advantage by raising the abstraction level of the underlying IoT systems to ease the developers task without sacrificing much flexibility. This is because once the underlying hardware, protocols and functionality of IoT systems are abstracted as nodes in a flow, much of the design of the application logic is simplified to manipulating node connections and processing generated data. When the developer needs more flexibility or functionality than the current nodes offer, new special-purpose nodes can be developed and deployed, or nodes that support embedded script languages can be used to leverage features of the underlying system, and implement new protocols or functionality that does not exist in the current system.

The primary aim of the study of the available frameworks and approaches presented above was to gain an understanding of the concepts and methodologies that can be used while developing an IoT application in a practical setup in fog computing environments. The frameworks such as Mobile Fog [74] and Foglets [86] use the concepts of modular design approach in their methodology. These are the first in line that present a move towards the modular design and programming approach in fog computing environments, thus also serving as a guiding direction for development of a further novel approach from practical standpoint.

Amongst the above, DDF appears to be the best choice while building an IoT application in fog enabled environments in a simulated or emulated test-bed setup. With

DDF, one can mathematically model the application as well as infrastructure into a graph representation and can perform various possible operations in simulation or emulation before moving to a real-world scenario. We later use the DDF approach in Chapter 4 for the application modelling in the iFogSim [62] simulation platform.

To our understating, the common consensus and output from the studies above is that ***an IoT application*** *should be able to execute over a group of devices to fully leverage the capability of these different devices in the infrastructure along the things-to-cloud continuum. Thus, it should ideally be designed as a collection of independent components.*

Accordingly, there is a need to have an approach for design and development of IoT applications that utilizes the computing resources available along things-to-cloud continuum, thus leveraging the fog computing paradigm and providing the functionality of the desired IoT application and use-case scenario. Part of the work presented in Chapter 3 is positioned as our contribution to fill this gap in literature.

Furthermore, there is also a need to provide a way to define the constraints that regulate where and how these individual components containing application logic (or part of application logic) should be deployed so as to efficiently exploit computation resources available. And that is where we position our work presented in Chapter 4 as the contribution in the literature to fill the gap.

2.7.2 IoT Application Domain: Smart Dairy Farming — Fog Computing Assisted Smart Dairy Farming

This sub-section presents the literature review performed in the selected application domain of smart dairy farming. It presents a collective literature review on the topics of IoT, fog computing and data analytics work in the selected application domain.

IoT, fog computing, cloud computing and data-driven techniques together offer a great opportunity for verticals such as dairy industry to increase productivity by getting actionable insights to improve farming practices, thereby increasing efficiency and yield. There has been active initiation and movement in the agricultural domain to move towards tech-enabled smart solutions to improve farming practices. The concept of Smart Dairy Farming is no longer just a futuristic concept, and has started to materialize as different fields such as machine learning have found a prosperous application in this domain. The section (2.7.2.1) below presents an overview of limits, opportunities and challenges in the field of agriculture in adopting smart solutions. Next, section 2.7.2.2 presents a comprehensive review of the works done using IoT, fog computing and data analytics in the field of agriculture, further progressing towards the limitations of those work and positioning our work to fill the gap.

2.7.2.1 Agriculture and ICT: Limits, Opportunities and Challenges

While there have been several approaches towards smart farming by an active incorporation of Information and Communication Technology (ICT) in the agricultural industry, a vast majority of them still remain practically unimplemented by virtue of the fact that a farm environment is much unlike an ideally 'connected' IoT use case. The true penetration of ICT in the most granular levels of global farming diaspora depends on the transformative adaptability of the IoT ecosystem, wherein a shift is required from a predominantly urban context towards a more flexible solution that addresses the unique environmental, temporal and spatial challenges of remote farm locations. The major limitations, challenges, and opportunities in the context of agriculture and ICT are described below:

1. Geographical Factors, Low Internet Connectivity and Weather Based Outages

What makes a farm scenario unique is the fact that typical farm locations are geographically remote from the urban context and attributed with sparse to low Internet connectivity. The existing IoT systems and applications that address certain specified objectives in the diverse agricultural domain largely depend on Internet connectivity for proper functioning, as the service and application components are traditionally deployed in a cloud-centric manner in remote cloud infrastructures.
Owing to their typical geographical locations, a vast majority of farms already have limited cellular [91] coverage. In that terrain, it is not uncommon to be facing long outages in Internet connectivity due to constraints posed by adverse weather conditions, storms, hurricanes, and other natural disasters. All of these scenarios are apart from general network downtimes, which might be because of another variety of reasons such as network maintenance, broken links, faults, and network/security attacks.

2. Cloud Based Solutions

The existing solutions suggested by industry and academia primarily include a lot of closed form cloud based solutions that lack flexibility in the management of data, and customization owing to on-demand services and operations. They often tend to get expensive, and the trade-off between cost and system utility are not wholly profitable to a small scale farmer.

3. Real Time Analytics

One of the key limitations of the existing solutions is the lack of support for real time processing and analytics for latency sensitive use cases. For example, while dealing with dairy cattle as in our case, a detailed analysis of patterns from the collected data would

enable the development of algorithms, which, once deployed on the fog node closer to the data source, would be able to identify latency critical scenarios in real time. The existing systems of historical, cloud based analytics are incapable of serving a time critical situation, and the bouts of no connectivity further worsen this management.

4. End-to-End Solution and Vendor Lock-In

Most of the solutions only focus on one tier of the problem, i.e., either data sensing, data analytics or consumer relations [92]. Architecturally, there is also a major issue of vendor lock in, wherein devices and applications of multiple vendor systems are incompatible with each other, and the lack of features within one solution cannot be complemented with its integration with another.

5. WiFi Sensors: Cost and Operational Trade-off

Not only is reading data and supporting sensors with WiFi capability challenging in such a remote scenario, but this also comes at a higher trade-off with the cost. Supporting sensors with an Internet connection hikes their price for both the acquisition and as well as in terms of operational expenses, while Internet connectivity in itself is a challenge in remote regions of operation.
A farm environment thus poses an atypical use case with heterogeneous demands, and the lack of a flexible solution impacts the acceptance of ICT towards what can otherwise be a very fruitful and profitable venture for both the farmer and the solution provider.

6. Existing Solutions and Challenges

There have been several propositions towards enabling connectivity on a farm scenario, as listed below:

(a) Microwave Wireless Terrestrial Link: A point to point connection, this technology is used by Internet Service Providers (ISPs) and cellular carriers to connect remote regions towards a backhaul network for last mile connectivity. This link can connect an ISP to a farm, which can then be distributed locally via WiFi access points. But this has its own sets of challenges and dependencies, and is more dependent on Internet providers rather than the individual farmer. Terrain and physical obstacles also impact such a connectivity, as does rain fade and other weather based factors.

(b) TV White Spaces : To enable connectivity on the farm, solutions such as FarmBeats [93] have proposed the use of unlicensed TV White Spaces to set up a link from the farmer's home Internet connection to an IoT base station on the farm, that further

provides a WiFi front end. However, this involves its own set of challenges and expenditures that might make such a solution difficult for adoption by a small scale farmer. For instance, the IoT base station needs power to work, which in itself is a problem in certain areas. One can use solar electricity, but that is highly reliant on weather conditions, and not reliable at this stage, considering the fact that no power on the IoT base station means no WiFi connectivity, and in turn, no data collection and transmission from sensors deployed.

This significantly reduces the reliance and adaptability of ICT on a farm, and is not dependable in case of an adverse scenario. Also, currently being unlicensed, it requires permission and authorization from Government and Federal agencies which is an additional overhead; thus making it difficult to be used in a number of countries at the moment.

(c) Data Mules : This is one such proposition that addresses the issue above in certain use-cases where moving objects such as vehicles or people serve as a mule to collect data from the sensors and upload it to the cloud as soon as Internet connection becomes available. It was initially proposed for the purpose of delivering emails to remote, rural or economically under-served regions [94] which suffer from the Internet connectivity challenge as specified above. The idea is not straight forward applicable and feasible with large scale sensor deployments as in such scenarios the storage capacity of micro-computers or on-chip computers available on sensor devices might get overwhelmed and over-utilized with that amount of data. Furthermore, even when the Internet connection is available, the bandwidth available could still be very low, thereby making the upload of large amount of data (in order of Megabytes or Gigabytes) coming from wide scale deployments of data infeasible.

2.7.2.2 IoT, Fog Computing and Data Analytics in Agriculture Domain

With the recent advancements in IoT, the use of computing systems utilizing wireless sensor networks (WSN) has been widely proposed in the agricultural sector in order to facilitate real-time monitoring of farm processes. IoT is an active enabler of smart farming, whereby various entities on the farm can be connected for collecting and exchanging data, thus allowing joint or independent operations. As technology grows to be an integral part of the agriculture and dairy industry, it is important to generate timely insights from the data collected, and enable effective data management.

There have been proposed systems in industry [95], [96], [97] as well as in academia [98], [99], [100], [101] for animal health management in dairy farms. A study by authors in [102, 103] gives an overview of the sensor systems available for health monitoring of animals in dairy farms. Along with the proposed systems and solutions, there has also been

research in integrating technologies to increase productivity and sustainable growth in agriculture; ranging from improving wireless network connectivity in economically under privileged areas [104], [105],[94], to data mining and analytics for agricultural applications [106], [107], [108] to providing decision control in variable environments under constraints [109], [110], [111]. A review by the authors in [112] shows that predictive insights in farming operations drive real-time operational decisions, and redesign business processes for the benefit of various stakeholders in a farming landscape, and that the influence of such systems goes beyond primary production, to the entire supply chain.

Authors in [113] provide a detailed survey of IoT enabling technologies that can offer automation, data aggregation and protocol adaptation in the wide field of IoT. They also present the required integration of IoT with emerging technologies such as data analytics and fog computing. Another survey in [114] identifies a serious lack of analytics and intelligence in existing smart dairy farming systems, thus leading to gaps between the desired requirement of the system and proposed solutions. It articulates the pressing requirement of intelligence to be present on the premises, in the on-farm systems.

As a consequence, attention is being drawn towards designing systems with intelligence and data analytics capability being present on premises [115], and utilizing fog computing comes to shore with those objectives in mind. While fog computing was initially aimed to cater to latency-sensitive, time critical applications and use-case scenarios, we believe that fog computing will act as an active enabler for computing systems, applications and services which suffer from constrained and unreliable cloud connectivity.

In our case, considering that the farm locations are usually remote with intermittent Internet connectivity, dependence on the cloud for pervasive monitoring and computation is less reliant due to a variety of connectivity issues, and other associated factors like high response latency, and high bandwidth requirements [116]. In such situations, fog computing can ensure high availability, and less reliance on remote cloud infrastructures. Such a system becomes even of more importance with adverse weather conditions such as during storms and hurricanes [117], [118], when the Internet services get interrupted and affected for various reasons.

Authors in [119] present the use of Raspberry Pis [120] as edge devices which are further connected with the cloud to demonstrate a smart farm computing systems for animal welfare monitoring. Authors demonstrated that a low-cost and open computing and sensing system can effectively monitor multiple parameters related to animal welfare. While animal welfare remains a broad concept, their paper shows that many parameters relevant to various stakeholders can be measured, collected, evaluated and shared, opening up new possibilities to improve animal welfare and foster high-tech innovations in this sector.

The authors in [121] propose the use of fog computing for innovative service creations for existing cloud based agriculture system. By means of simulation, the authors demonstrate that fog computing presents a unique capability for a creative IoT platform adoption in agriculture with existing cloud support. Quite recently, authors in [122] describe the design, development and evaluation of a system that covers extreme precision agriculture requirements by using automation, IoT technologies, and edge and cloud computing through virtualisation.

Authors in [123] present the need of data driven movement in agriculture in order to improve crop yields, improve quality, and reduce costs. Another recent survey by authors in [124] identifies the lack of interoperability provided by such systems, and the need of developing an integrated system combining edge, fog and cloud to provide application and services. The authors here also identify that technology solutions with no consideration of interoperability results in vendor lock-in, which not only hinders innovation, but also results in higher costs for the farmer/user.

One of the primary limitations of the previously proposed systems is that they follow historical data analysis and perform only cloud based analytics without leveraging and efficiently utilizing the resources [10] available on the farm along the things-to-cloud continuum [12]. Moreover, such techniques are not always suitable for real-time tracking and monitoring of dynamic entities such as dairy cows. The gaps with the existing research is that either it has been developed out of the agricultural context, or addresses the issue of analytics and control in isolation; this has also been identified as key limitations by authors in [125].

While there has been an initiated movement towards data-driven agriculture in recent times for sustainable and productive growth, there is still lack of leveraging emerging paradigms such as fog computing, and applying innovating machine learning models to solve a specific problem in the dairy sector. Most of the articles in literature present results based on simulated experiments, and those which come from real world deployment are mostly agriculture based, and rarely based on dairy farming. Further, only some of them have a machine learning element to automate their approach. However, to the best of our knowledge, no prior work focuses on providing an end-to-end IoT solution integrating edge, fog and cloud intelligence specifically in case of smart dairy farming IoT settings.

We position our work as an answer to the issues mentioned above, thus bridging the gap, and providing an innovative way that integrates edge, fog, cloud computing and machine learning to provide a solution specifically in case of smart dairy farming in an IoT setup. The novelty of the proposed approach comes from the standpoint that it has been specifically designed and developed to address a specific vertical of the IoT ecosystem i.e., dairy farming, and within that to address a specific problem related to animal welfare i.e., detecting lameness at an early stage before the clinical signs of it appear, and the

microservices oriented design makes it multi-vendor interoperable. This has been detailed in Chapter 3.

2.7.3 IoT Application Placement in Fog Computing Environments

It is clear from the discussion and studies presented in section 2.5 and 2.7.1 that, an IoT application should be built as a collection of components. This brings us to the next question of efficient deployment of these components on the available computing resources in the infrastructure.

Deploying all computational components of an application in the remote cloud is not always necessary and effective. We study how to distribute and deploy these components of a multi-component application between fog and cloud, and effectively utilize the computational resources available in the infrastructure.

What is application placement? The application placement[2] defines a mapping pattern by which application components are mapped onto the computing devices available in the infrastructure. Fig. 2.11(b) [126], [127] shows an example of a cognitive assistance IoT application modelled as a Directed Acyclic Graph (DAG) (Fig. 2.11(a)) mapped onto the available computing devices in the infrastructure [127].

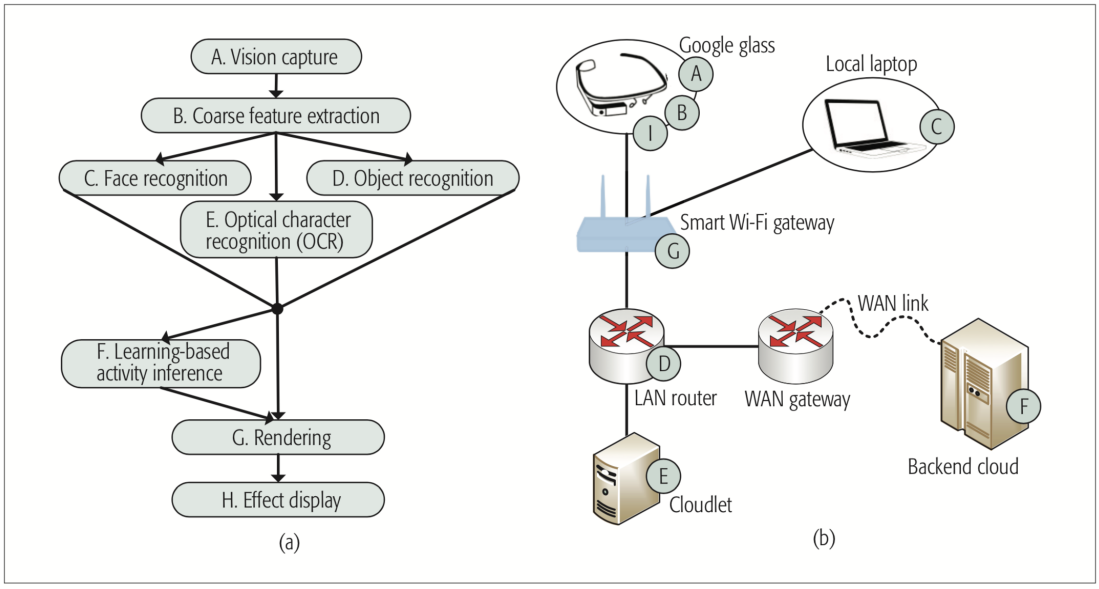


Fig. 2.11 Directed Acyclic Graph (DAG) of an application and its deployment onto the infrastructure [126], [127]. (a) DAG of a cognitive assistance application, (b) Deployment of this DAG onto the computing resources available in the infrastructure.

[2]It should be noted that we use application placement and application deployment terminology interchangeably in this dissertation. Within the scope of this document, they both refer to the same thing.

The DAG is the most often used representation of a multi-component application as it models and covers a large range of realistic IoT applications like video processing [128], [129], gaming[130], healthcare [131] etc. The application placement involves finding the available resources in the infrastructure that satisfies the application requirements and the constraints, and optimizes the objective functions (if any). Service providers have to take into account these constraints to first, limit the search space, and second, to provide an optimum or near optimum placement. It should be noted that DAG is a special topological representation of a connected graph, and other possible topological representations include line graph, tree application graph, etc. But as mentioned above, the DAG representation covers a large range of realistic IoT applications as suggested by literature; hence we consider the DAG representation in our study for application placement research.

Mathematical definition of application placement: Let G be a multi-component application with a set of requirements R, and let N be the set of computing resources available in the infrastructure. Solutions to the application placement problem are mappings of each component of G to some computational node in N, meeting all the requirements set by R, and also optimizing objective function(s), if any.

When deciding on where to deploy application components over the continuum from things-to-cloud, application administrator or service providers need to find the best deployment that satisfies all the application requirements over the available resources in the infrastructure along this continuum , and also optimizing objective function(s) if any.

2.7.3.1 Application Placement Approaches in Literature

Modern day applications are not monolithic anymore [132]. Therefore, an application running in a fog computing environment consists of a set of independently deployable components (or services, or microservices) that work together to meet some objective.

How to deploy multi-component applications? — The problem of multi-component application(s) placement has been thoroughly studied in the cloud computing scenarios. The work towards application placement in fog computing environments as presented in this dissertation was was one of the early developments in this research direction at the time. A lot of inspiration was drawn from the work available in the cloud computing and related areas. Over here, we systematically present the most related work and the literature studies from whom the work presented was inspired, and also highlight the novelty of our work within the scope of the timeline when this part of research was performed.

Cloud Computing Application Placement Work: The application deployment problem in cloud infrastructure has been studied by various researchers. Authors in [133] defined a process algebraic approach named cloud calculus to specify deployment, migration and security policies of virtual machines (VMs) across different clouds. Based on

a cloud computing network architecture, virtual machine replication and a complementary merging mechanisms were proposed and analyzed by authors in [134], capable of exploiting the locally available information to reduce the communication and support cost in the network. Authors in [135] proposed a service placement architecture for IoT based on Integer Linear Programming (ILP) that continuously adapts services according to the changing network conditions and user status. A model-driven optimised planning solution to deploy software applications in cloud /multi-cloud environment was proposed in [136] and [137]. Likewise, solutions such as [138], [139] for automated application deployment in cloud or multi-cloud environments are in regular use by the DevOps community . An evaluation of a set of heuristic algorithms for solving the service placement problem in the context of computer networks has been presented by authors in [140]. It minimizes the end-to-end delay, and elaborates a layered graph placement algorithm that proposes to find a lowest cost path that includes communication cost and processing cost. It was proposed by authors for the next-generation network architecture vision at that time, but it does not consider the fog computing environment and related resource constraints.

With respect to the work available in cloud computing domain, fog computing introduces the challenges of dynamicity, distributed and heterogeneous computational resource environment with IoT, which were not taken into account by the research at the time.

Fog Computing Application Placement Work: Among the first investigations in this direction was the work presented by authors in [62], where they proposed an edgeward placement algorithm that determines an eligible deployment of multi-component applications modelled as DAG in fog computing environments. The authors here released an open-source simulator iFogSim, which to best of our knowledge was the first simulator at that time specifically developed for simulating fog computing environments to be released in academic and research community. It is developed over the CloudSim [141] framework for modelling and simulating IoT, edge and fog Computing environments. Building upon this work along the same line, we developed a heuristic based application placement algorithm presented in chapter 4, exploiting binary search as heuristic to find the best placement solution for a test application.

A heuristic algorithm to solve module deployment problem in fog computing has been proposed by authors in [142]. The developed heuristics maximizes the number of satisfied services, and more specifically, it limits the maximum number of modules that can be deployed on a device. However, it does not consider the resource requirements of the modules during deployment as considered by our approach for efficient resource utilization.

Authors in [143] presented an Integer Linear Programming (ILP) formulation for the QoS-aware service allocation problem for combined fog-cloud scenarios. They minimize the service latencies in fog while satisfying the QoS requirement. Authors in [144] pro-

posed a solution for the distributed data stream application placement in a geographically distributed environment with the goal of minimising the end-to-end application latency.

To the best of our knowledge, when this part of the work looking into the deployment stage of IoT application life cycle in fog computing environments was performed, no approach other than [62] and others presented above was available that looked into this, and there was no existing work specifically proposed for application deployment in fog computing environments.

2.7.4 Distributed Decomposed Data Analytics in Fog Computing Environments

In this section we first present a brief background of data analytics in fog computing environments (2.7.4.1), followed by a comprehensive view of data collection pipeline in such deployments (2.7.4.2), which maps with the IoT data life cycle presented in Chapter 1 (Fig. 1.1(b)). We present the areas of further improvement with fog computing into the picture in this data collection pipeline of the IoT data life cycle, which sets up the motivation behind the proposed distributed decomposed data analytics approach in fog computing environments. Section 2.7.4.3 presents the literature review of the work done in this direction, and positions the work presented in the dissertation to fill the gap.

2.7.4.1 Data Analytics in Fog Enabled IoT Environments

While IoT deployments vary across use cases, the most prominently common underlying aim is to analyse the data generated from the devices to achieve a specified objective. Majority of current IoT data processing solutions transfer the data to the cloud for processing. This is mainly because existing data analytics approaches are designed in a cloud-centric methodology. With millions of IoT devices generating data, transferring all of that to the cloud is neither scalable nor suitable for real-time decision making. As data travels from its point of origin (e.g., sensors) towards applications deployed in cloud virtual machines, it passes through many devices, each of which is a potential site for computation offloading. Therefore, it is important to take advantage of computational and storage capabilities of these intermediate devices.

Analysing data during the early stages in the infrastructure pipeline presents additional benefits of data and communication security in the overall system, owing to the fact that raw data is now processed closer to the data source, and only processed data is sent further. The efficient use of available computational resources realized by means of fog computing, in turn, promotes the idea of green computing [145] as well. Fog nodes have limited computational capabilities. In such circumstances, one of the requirements is

to have independent data analytical components that can be remotely pushed onto fog devices, such as module-based. These modules would be black boxes running in individual sandboxes where they intake certain types of inputs and generate certain types of outputs. Performing data analytics or partial data analytics operation on data at early stages of infrastructure pipeline may help in reducing the amount of data being sent to nodes further in the hierarchy leading to savings in data communication cost, storage and computational cost.

2.7.4.2 Data Analytics Decomposition in Fog Enabled Environments

With IoT generating large volumes of data, transferring data from sensors to remote data centers is currently not efficient from a performance perspective due to limitations on bandwidth and high latency. With data being generated and collected in a distributed manner, it becomes necessary to look for approaches where knowledge extraction from this data can also be done in a distributed manner. A comprehensive view of the data pipeline is presented in Fig. 2.12, with Fig. 2.13 highlighting the areas with a scope of improvement by the incorporation of fog computing.

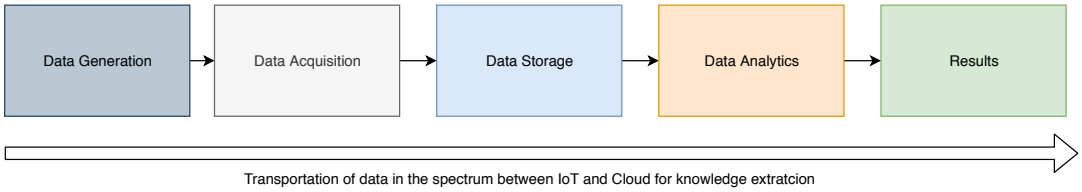


Fig. 2.12 A comprehensive view of data collection pipeline.

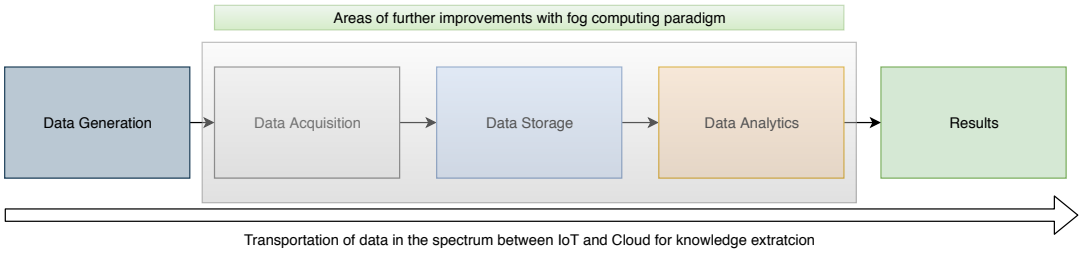


Fig. 2.13 A comprehensive view of data collection pipeline and highlighting the areas with a scope of improvement by incorporation of the fog computing paradigm.

With fog computing into picture, in this pipeline, the collection approaches can be changed and the data analytics processing algorithms can be re-structured to perform operations closer to the data source. The classic data reduction techniques such as filtering, selective forwarding, summarizing etc. all apply, but now along with that, these fog nodes at the edge of the network can be used to perform data analytics operation on them.

However, these fog nodes being resource constrained in nature might not be able to run the full analytics service, so the idea is that if the device cannot hold the full service, then only put the part of the service that can be handled by it. Thus, the approach of decomposing the computing program into smaller units seems like an ideal proposition in fog enabled IoT environments. A detailed literature review related to data analytics decomposition within the scope of work presented in this dissertation has further been presented in the next section 2.7.4.3.

2.7.4.3 Literature Review of Related Data Analytics Approaches in Fog Computing Environments

In traditional centralized approaches [146], [147] for data analytics in IoT, all collected data are transferred to centralized location such as server(s) in data centre (i.e., the cloud) and is then subjected to the desired data analytics model; thus such approaches suffer from the bottleneck of data transfer. However, when the data itself is generated in a distributed manner (as with IoT deployments), transferring it all to the centralized cloud is an additional overhead when it can actually be processed in a distributed manner as well. Furthermore, in some use-cases such as healthcare, network nodes might not want to share data because of privacy issues.

There are approaches [148], [149] that have been proposed for Wireless Sensor Networks (WSNs) based on selective forwarding that take into account the constraint of bandwidth, latency and energy. The issue with such approaches is that they only focus on communication efficiency without being aware of analytical task being performed at the destination. Further advanced methods based on selective forwarding [150], [151], [152] work on dynamic optimal decision making to find best time to deliver data for communication efficiency and to minimize reconstruction error at the destination. However, such methods are limited to communication overhead and have not be developed and applied to the network edge i.e., in a fog based setting.

Authors in [153] present the survey of a subset of methods that can be be modified to run in a distributed manner to solve the problem of linear least-squares. Distributed approaches [154], [155], [156] specific to regression work on the constraint that gathering data centrally is either expensive or impossible, and focus on distributing estimation of global model parameters over nodes with the aim to achieve the same prediction performance that would have been achieved by the corresponding centralized model. The issue with such approaches is that they need additional techniques for parameters update and synchronization, which restricts their use for a wide set of IoT based applications.

Recent work [157], [156] in edge-analytics exploits the computational power of devices such as Raspberry Pi [120] and BeagleBone [158] to design and launch lightweight

algorithms directly at the data sources. Authors in [157] present edge stochastic gradient descent (EdgeSGD) algorithm for solving linear regression problem with the objective of estimating the feature vector on the edge node. They show that such approaches can converge faster to the optimal values as compared to the centralized approach. Their approach is different from ours as their approach is iterative in nature and needs to converge to find the solution, while we present the closed form solution of the problem that fits to the parameter without the need to use an iterative algorithm. We elaborate more on this later in chapter 5 (section 5.3).

Authors in [159] and [160] present methods of data suppression based on local forecasting models on sensors with the aim of re-constructing data at the sink node. These methods exclusively focus on reducing data communication by means of data suppression using forecasting models.

Initial exploratory work by authors in [161] shows that such decompositions can reduce bandwidth consumption and can significantly decrease the associated costs. But further research and developments on areas like decomposition methods, system performance and quality of analytics need to be carefully studied to design efficient distributed solutions for fog enabled IoT settings, and that is where we position our work.

Our work presented in Chapter 5 decomposes the desired data analytics model to run on the edge of the network coping with the above mentioned constraints. Our approach, as devised from [162] and explained further in in Chapter 5 (section 5.3) does not modify the algorithm in use, rather remains an exact implementation of it albeit capable of running in a distributed manner in fog enabled IoT deployments.

2.8 Summary

This chapter presented a review of the technologies, trends and current literature that are supplemental to the research proposed in this dissertation, and laid a foundation for the work done. It commenced with a background study on IoT, IoT architecture, and limitations of a cloud-centric approach, leading to an evolution of fog computing, its associated concepts, definitions, terminologies and challenges that come with these paradigms that need to be addressed. Next, a literature review in the contribution areas of the thesis was presented. These are all related work that help in identifying the contributions made through this research work.

As presented in the chapter, most existing research has been fragmented when it comes to the decision making processes related to IoT application life cycle management in fog computing environments. Either the work has been performed in isolation without leading to any practicality from it or it has been done out of context of fog computing environments.

To the best of our knowledge, limited attempts have been made in the combined research direction of IoT, IoT applications and fog computing as presented in this dissertation.

Our work presented in this dissertation focuses on design, development (**Research Question 1**), deployment (**Research Question 2**) and data analytics functionality (**Research Question 3**) of the IoT applications in fog computing environment; thus making contributions in improving the decision making process related to the associated stages of IoT application life cycle management. This chapter highlighted how current literature is unable to address the research questions defined for this dissertation. It also helped to compare and contrast between the proposed research and existing work.

In summary, the dissertation presents – **how** to develop fog enabled software system in an IoT oriented use-case? **What** software design methodologies are appropriate to use? **What** is the efficient deployment i.e.,**where** to place the application component(s) that need to perform computation? With extracting knowledge from data being prime objective in IoT deployments, **can** data analytics be decomposed to make it easier to run on resource constrained devices? **How** to do that? **What** approaches or methodologies can be used to do that?

Chapter 3

An IoT Application Design and Development in a Real-World Smart Dairy Farming Scenario Leveraging the Fog Computing Paradigm

3.1 Introduction

The Internet of Things (IoT) is about connecting people, processes, data, and things, and is changing the way we monitor and interact with them. An active incorporation of information and communication technology coupled with sophisticated data analytics approaches has the potential to transform some of the oldest industries in the world. It presents a great opportunity for verticals such as the dairy industry to increase productivity by getting actionable insights to improve farming practices, thereby increasing efficiency and yield. Dairy farms have all the constraints of a modern business — they have a fixed production capacity, a herd to manage, expensive farm labor, and other varied farm-related processes to take care of. In this technology-driven era, farmers look for assistance from smart solutions to increase profitability and to help manage their farms well.

In this chapter, we present the design and development of an IoT application in a real world smart dairy farm setup. We outline the key design principles and methodologies used in the development of the application. The developed IoT solution uses a distributed modular application architecture using microservices that allows the developed software solution to leverage the fog computing paradigm in such IoT deployments. The presented solution is an end-to-end IoT application system with fog assistance and cloud support that analyzes data generated from wearables on cows' feet to detect anomalies in animal behavior that relate to illness such as lameness. The solution leverages behavioral analytics

to generate early alerts toward the animals' well being, thus assisting the farmer in livestock monitoring. This in turn also helps in increasing productivity and milk yield by identifying potential diseases early on. The application developed specializes in detecting lameness in dairy cattle at an early stage, before visible signs appear to the farmer or an animal expert. Our trial results in a real-world smart dairy farm setup, consisting of a dairy herd of 150 cows in Ireland, demonstrate that the designed system delivers a lameness detection alert up to three days in advance of manual observation with an overall accuracy of 87%. This means that the animal can either be isolated or treated immediately to avoid any further effects of lameness. Moreover, with fog based computational assistance in the setup, we see an 84% reduction in amount of data transferred to the cloud as compared to the conventional cloud based approach. Further, the proposed application design and development methodology of having a hybrid modular-microservices approach, coupled with the fog computing paradigm make the solution scalable and fault tolerant as well.

The research project was termed **SmartHerd**, with a time-line from year 2016 to year 2018. Hence, the overall solution and software developed is referred to as **'SmartHerd'**, and the fog node in use is termed as **'SmartHerd IoT Gateway'**. A high-level view of system workflow and different components of the developed IoT solution are presented in Fig. 3.1.

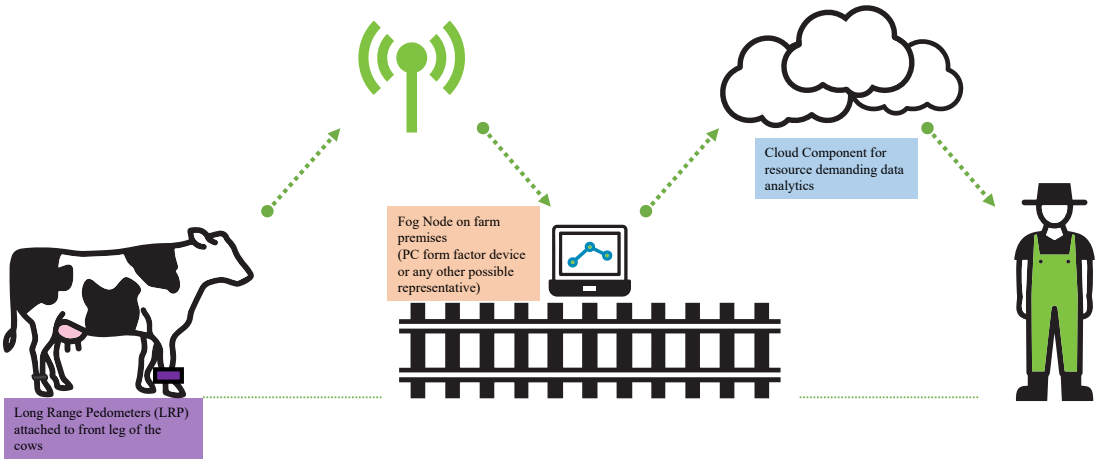


Fig. 3.1 A high-level diagrammatic representation of the system workflow, and different components of the developed IoT solution [8].

The **SmartHerd** project had the smart dairy farm deployment in Ireland with a full dairy herd of **150** cows, with a single vendor as the wearable sensor providers for the cows. Moreover, to further validate the proposed approach for early lameness detection, we are expanding the work undertaken in SmartHerd to date through the execution of a

use case in the IoF2020 project[1] named **MELD**[2]. This follow-on project is building and expanding upon our existing work presented in this chapter, and integrating it into the IoF2020 dairy farming technology trials with deployments in Ireland, Portugal, Israel and South Africa. It leverages sensor technologies from two different vendors on a combined total of approximately **1000** cattle, consisting of both beef and dairy.

This chapter is structured as follows: §3.2 describes the problem, research question being addressed, and also lists the sub-questions that were answered as part of this work, §3.3 presents the specified objective of the IoT solution, §3.4 presents the experimental setup and the real-word test-bed deployment done, §3.5 presents the proposed design and development methodology for fog enabled software deployment, §3.6 presents the microservices oriented application design and architecture of the developed solution, §3.7 presents the results, output from the validation experiments, discussion and analysis of results obtained, and finally §3.8 summarizes and concludes the chapter.

The work presented in this chapter has been disseminated in the following publications: **P4 - WF-IoT 2018** [5], **P5 - CCNC 2019** [6], **P7 - SPE 2019** [7], **P9 - IEEE IoT Magazine 2019** [8] and **P10 - COMPAG 2020** [9].

3.2 Problem

The first research question (RQ1) is being addressed in this chapter: **How to design and develop an IoT application leveraging the fog computing paradigm?**

Specific to the selected application domain of smart dairy farming, and the particular use-case objective of early lameness detection, the question can be phrased as below:

How to design and develop an IoT application with a specified objective in a smart dairy farm scenario leveraging the fog computing paradigm?

The above research question has been answered as a major contribution in this work. Furthermore, we also address the integrated sub-questions that arose as a part of this. These can be divided into two parts — one from fog computing and software development perspective, and the second is data analytics and machine learning element of the developed solution. A brief description of them have been provided below:

A.) Fog computing and software development: The integrated questions in this part consist of the system and application architecture understanding presented in Chapter 2, section 2.5.

1. *Which network device among the available options along things-to-cloud continuum should be leveraged as a fog node in such IoT deployments?*

[1] Internet of Food & Farm 2020, https://www.iof2020.eu/

[2] MELD stands for **M**achine Learning based **E**arly **L**ameness **D**etection in Beef and Dairy Cattle

Fog computing is an emerging computation paradigm that aims to extend cloud computing services to the edge of the network, thus enabling computation closer to the source of data. It has been used increasingly in IoT applications, especially in constrained network and Internet connectivity scenarios, which is also one of the issues in remote farm-based deployment such as ours.

Most IoT enabled smart farms have some sort of farm management system in place which usually runs on a PC form factor device available within farm premises. Farmers use it to maintain logs and to keep other details electronically at hand. So our plan was to utilize the computing resources already available in such scenarios and leverage them under the fog computing paradigm. Thus, we chose the laptop available with farmer in our case as the fog node. It should be noted that the developed system is fully capable to adapt if the fog node is changed to any other possible representation such as a gateway device. A detailed discussion on this has been presented later in this chapter in section 3.7.1 (§3.7.1.4).

This decision also helps to improve fault tolerance, and build up the system resilience to variable farm environments such as weather-based network outages and connectivity issues owing to geographical remote locations of farms. In scenarios with low/no Internet connectivity, it becomes ideal to process the data locally as much as possible, and send the aggregated or partial outputs over the Internet to the cloud for further enhanced analytical results. The fog computing based approach leads to effective utilization of the limited bandwidth available, and reduces the dependency on the cloud by facilitating a part of data analytics involved in the solution on the network edge.

2. *What should be the development design of the system so that it should be usable, compatible and able to serve in both user possible scenarios listed below:*

   (a) when a farmer acts as the end-user?

   (b) when an agri-tech service provider acts as the end-user?

   The end user in our scenario could be a farmer with an existing system or an agri-tech service provider who wants to provide more services to their clients. With that in mind, we decided that the system should be developed as 'Application/Software as a Service'(AaaS/SaaS), which can be used by the service providers to integrate with their existing systems, or can be used directly by the farmer as well.

3. *Which software development methodology to use so that the designed system should be multi-vendor inter-operable, and also be in-line with the finalized design of question 2 above?*

   The answer and discussion on this has been presented in detail in section 3.5.

B.) Data analytics and machine learning: The sub-questions here in this part deal with the specified objective of the IoT solution being built and mapped to the IoT data life cycle of the underlying deployment.

1. **Cow Profiles:** *How to build robust cow profiles that are distinguishable by the learning model as lame and non-lame? Which parameter to use as baseline while building and comparing cow profiles?*

2. **Clustering:** *Does each animal in the herd need to be treated separately i.e., treating each cow as a single experimental unit; or can some clustering technique be used to define clusters of animals that share similar features within the herd?*

3. **Classification — Early Lameness Detection:** *Which classification model to use given the objective of early detection of lameness in dairy cattle?*

The answer and discussion on these three questions of part 3.2 -**B** have been presented in detail in the section 3.7.3.

3.3 Objective of the Developed IoT Solution — Early Detection of Lameness in Dairy Cattle

Dairy farmers work hard from dawn till late in the evening — milking, feeding and maintaining the farm. So, it is a challenge to monitor the well-being of hundreds of cows in a dairy farm in real time. The methods for looking after animal welfare are based on millennia of human experience, and grounded on observational methods to analyse animal behaviour by visual observation for some kind of anomaly or potential health-issue. This leads to the question — *Could technology help? Can there be a better way to do it?*

There are behavioural changes when animals become ill, which can be mapped to specific illnesses. The risk of diseases has a large effect on the economy of a farm — payment for veterinary treatments and loss of milk production from the infected animals, as well as on animal welfare. *So, what if one could detect the onset of common diseases before any symptoms are even visible?*

To reiterate, the health and welfare of dairy cows is paramount to the productivity of the herd in both operational and capital expenditure related to pasture management and milk production. One of the issues that need to be addressed in this domain is lameness management.

Lameness in Dairy Cattle: Lameness is a condition that affects the locomotion patterns of livestock. An all-encompassing definition of lameness includes any abnormality which causes a cow to change the way that she walks; and can be caused by a range of foot

and leg conditions triggered by disease, management or environmental factors. Controlling lameness is a crucial welfare issue, and is increasingly an inclusion in welfare assurance schemes.

Lameness is considered to be the third disease of economic importance in dairy cows after reduced fertility and mastitis [163]. It is estimated [164] that lameness costs an average of €275 in treatment per instance. Early and timely lameness detection allows farmers to intervene earlier, leading to prevention of antibiotic administration and improvement in the milk yield, as well as saving on veterinary treatment for their herd.

The existing solutions for lameness detection in dairy cattle either have high initial setup costs and complex equipment, or, in the ones that are technology based, major interoperability issues towards compatibility with existing farm based management solutions. As a solution to this, we have developed an end-to-end IoT application that leverages machine learning and data analytics techniques to monitor the herd in real-time, and identify lame cattle at an early stage.

3.4 Experimental Setup — Real World Test-bed Deployment

Focused on animal welfare and health monitoring, this deployment involves installing sensors on cows' feet. Data generated from these sensors is subjected to analysis using fog computing, which is further enhanced by its cloud component that acts as the site for data fusion and other related resource demanding data analytics functionalities.

3.4.1 Smart Dairy Farm Setup

The trial was conducted on a local farm with a full dairy herd of 150 cows in Waterford, Ireland. It should be mentioned here that the ethical approval for the experimentation was taken from Research Ethics Committee of Waterford Institute of Technology, Ireland prior to the deployment in July, 2017.

From the options available for the sensors/wearable for livestock monitoring, we decided to use radio communication based Long Range Pedometer (433MHz, ISM band) instead of WiFi based. The reason behind this was that the former does not depend on the Internet for its operation, and serves the purpose of data acquisition in farms where network connectivity is a constraint.

These wearables have less operational expense, and do not use WiFi based connectivity to send sensed data to a base station. Therefore, as a part of the real-world deployment, off-the-shelf available Long-Range Pedometers (LRP, ENGS Systems©®, Israel) specially

Fig. 3.2 Cows with long-range pedometers (LRPs) attached on one of their front legs as part of our smart dairy farm setup [5], [7], [9].

designed for livestock monitoring were attached to the front leg of cows as shown in Fig. 3.2.

3.4.2 Real World Test-bed Architecture and System Overview

The overall architecture of the test-bed is shown in Fig. 3.3. The pedometer consists of an active system with a (backup) data retention capacity of upto 12 hours that measures the activity of cows (standing, lying, walking, etc.) with a sampling frequency of 8 milliseconds; and the data unit thereby generated is sent to the corresponding receiver and transceiver every 6 minutes. The range of the antennas attached to the receiver and transceiver is 2 kilometres each, which gives enough coverage to collect data from cows at all times, whether they are grazing in the field, present in their sheds (during adverse weather conditions), or being milked at the milking station.

As shown in Fig. 3.3, the receiver is the master unit which sends the received data to the communication unit (RS485 to USB) through wired connection, which in turn then

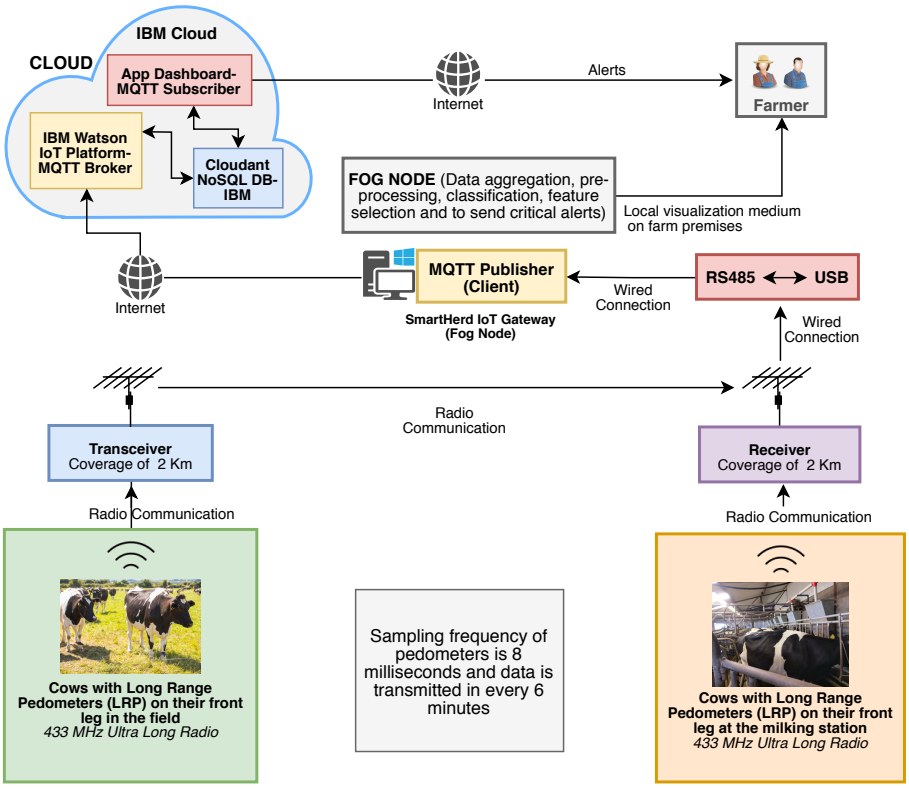


Fig. 3.3 SmartHerd Management system overview — overall architecture of the test-bed [5], [7], [6], [9].

sends it to the SmartHerd IoT gateway (a PC form factor device in our case, which acts as controller and fog node. The configuration used is Intel® Core™ 3rd Generation i7-3540M CPU @ 3.00GHz, 16.0 GB RAM, 500 GB local storage) through wired connection via a USB interface.

The fog node consists of a local database which stores all the raw data as received from the sensors. This is then pre-processed and aggregated their to form behavioural activities, and summed to form hourly and daily time series. In this study, we used the following three behavioural activities for the analysis:

1. *Step count:* This is the number of steps an animal takes.
2. *Lying time:* The number of hours an animal spends lying down.
3. *Swaps:* This is the number of times an animal moves from lying down to standing up.

The choice of these three parameters is based on literature survey, which suggests that these three act as the best predictors of a lame cow (or one transitioning to lameness) while analyzing movement or activity patterns of cows.

Connectivity protocol between fog node and cloud: There are a number of options available when it comes to streaming the data, for e.g. MQTT (Message Queue Telemetry Transport) [165], AMQP (Advanced Message Queuing Protocol) [166], XMPP (Extensible Messaging and Presence Protocol) [167], etc. Each of these have their individual pros and cons, and selecting one depends on the use-case, objective and IoT deployment scenario. Our aim was to use a lightweight protocol that can work in our use-case and is also widely supported by both academia and industry in such scenarios. After evaluating and comparing the available options, we selected MQTT as the connectivity protocol between fog node (i.e., local PC) and cloud (service instances running on IBM Cloud) in our deployment setting.

MQTT is an open-source protocol originally invented and developed by IBM [168]. It is a lightweight publish-subscriber model based protocol designed on top of the TCP/IP stack. It is specifically targeted for remote location connectivity with characteristically unreliable network environments such as high delays and low bandwidth [169], which is one of the issues in remote farm based deployments such as ours. Hence, we chose MQTT as the connectivity protocol in our deployment.

MQTT provides three QoS (Quality of Service) levels [170] that define the guarantee of delivery for a specific message between sender and receiver:

- At most once (QoS 0): There is no guarantee of delivery here. If the failure happens, no additional attempts are make to re-handle those messages. It is often called "fire and forget", and provides the same guarantee as the underlying TCP protocol.
- At least once (QoS 1): At least once means that messages in a stream are guaranteed to be delivered at least one time to the receiver. If the failure happens additional attempts will be made to re-handle those messages. This approach may cause unnecessary duplication of data packets in the streams.
- Exactly once (QoS 2): Exactly once means that messages are guaranteed to be handled exactly the same as it would be in the failure-free scenario, even in the event of various failures. It is the safest and slowest quality of service level.

A developer can specify the QoS level they want on both publisher and subscriber as per their application requirements. We used QoS 2 in our setting.

The MQTT architecture comprises of two components, namely MQTT clients (such as publishers and subscribers) and MQTT broker (for mediating messages between publishers and subscribers). In our setup these components are as follows:

- **MQTT Publisher:** Script running on fog node (i.e., SmartHerd IoT gateway)
- **MQTT Broker:** IBM Watson IoT Platform (as a service on IBM Cloud)

- **MQTT Subscriber:** Application designed and hosted on IBM Cloud

Thus, the data from fog node after processing as described above is streamed to IBM Waston IoT platform using MQTT; the IBM Watson IoT platform receives all these messages, and the MQTT subscriber listening to the events of this broker picks up all the data and stores it in Cloudant NoSQL JSON Database (Database service on IBM Cloud).

3.5 Designing and Developing Software Systems in Fog enabled IoT Environments with Cloud Support

Designing and developing software systems is an intricate process which requires profound understanding of the procedure, consideration of software architecture and development techniques involved, and knowledge of various interconnected components in the deployed physical or virtual infrastructure.

Microservices: Given its successful and wide adaptation in cloud computing domain, a microservices based architecture seems quite an obvious candidate for use in such fog enabled IoT deployments, but its use is not straightforward. The design and operational practice is sometimes quite different between these two technological paradigms [77]. The major reason for this can be that the microservices approach comes from a different perspective, which is to efficiently build and manage complex software systems, which in turn came to realization as a move towards architectural modularity. The main drivers of modularity are: agility, testability, deployability, scalability, and availability. The challenge now is how to apply the microservices software development approach to build the application in an IoT scenario leveraging the fog computing paradigm.

Based on the literature survey done as presented in Chapter 2 (section 2.7.1) , and other qualitative useful resources available on the Internet, we proposed a hybrid approach for designing and developing IoT applications following a distributed modular application architecture using microservices. We validated this approach in our application domain of smart dairy farming with the use-case objective of early detection of lameness in cattle using locomotion data.

In our analysis, we found that a distributed modular application architecture using microservices was the best approach, given we could align with the service-based and event-driven needs of our application.

The analysis leading to this has been presented below:

❖ **Domain Driven Design and Data Requirement Observation:** First, we followed a domain-driven and data-driven design approach for the application being developed. The first step was to identify what data are we collecting or can be collected from a smart

dairy farm IoT setup. And based on this data what kind of services we can provide to the end-user. We conducted survey of dairy farmers, dairy-tech service provider companies and also consulted the available literature. The output of this process has been presented in table 3.1 as a collection of different attributes. Insights on this came more from survey than from literature. This is in line with the domain driven design methodology of software development i.e., to work closely with a domain expert (majorly the farmer in our case) and gain a better understanding of how the real-world system currently works and what are the demands/needs there.

Table 3.1 Data requirements with sensitivity level and latency constraints of various application and services in a smart dairy farming environment.

Application/Service	Latency Constraints	Data Requirement
Query sensor data	Minutes	Immediate
Livestock location monitoring and mobility analysis	Minutes	Immediate
Heat detetcion	Minutes	Immediate
Lameness and other illness detection	Hours - Days	Non-immediate
Animal health statistics	Hours - Days	Non-immediate
Logging and other application performance logging services	Hours - Days	Non-immediate
Animal behavior and variability analysis via mathematical modeling	Days	Non-immediate

We make the key observation that data requirements and latency constraints of various applications and services in a smart dairy farming scenario can be primarily classified in the categories as shown in table 3.1. This along with other reasons mentioned in Chapter 2, section 2.7.1 motivates the use of microservices based architecture and fog computing based approach in smart dairy farm setup.

❖ **Service-based and Event-driven needs of the application:** The next part after the above analysis was to look into the service-based and event-driven need of the application. Service-based need means that to provide a service what kind of data or input(s) are required and who is the consumer of that service. This mainly aligns with the needs of the user. In our use-case, service-based needs of the application are as follows:

- A service that the farmer can use to see the locomotion pattern of a particular cow or the whole herd.
- A service that the farmer or an animal expert can use to annotate or add details for the whole herd or for a particular cow.
- A service using which the farmer wants to check whether a particular cow is lame or not, and thus a data-query service needs to be in place.

Event-driven need refers to the events that are possible in the application workflow, and the corresponding services that need to be triggered to handle those events. This aligns with the possible events that can happen in the system. For e.g., in our use case, finding out that an animal(s) is lame is an event, and the farmer needs to be informed of that. Other such possible events are to alert the farmer when an animal goes out of a specified geographical boundary, when an animal falls down, or when an animal is in heat, or if there are constantly inconsistent locomotion patterns for a particular animal.

Most of the IoT applications will have both of these needs, or at least one of the above. Combining the conceptual approach of modular design with practical microservices based approach gave us a way to align the approach with both of these. For e.g. following modular approach one can write a data-query and search function which can be used by any other service needing this functionality and logic to generate the output. It is not necessary to create such search functionality as a separate microservice, rather one can build it as modular part of the code-base accessible to all services.

Modularity is a must, though not every portion of production has to be a microservice. Microservices need collaboration, and only when there are one or more drivers present should one make use of microservices. In the our use case scenario, we had all of the drivers for having modularity in the solution developed i.e., agility, testability, deployability, scalability, and availability. Microservices come with a set of advantages that make it an ideal architectural style for software development in end-to-end IoT solutions with constrained environments, giving the ability to overcome the constraints of vendor lock-in, while attributing technological independence between each set of services that make up an application.

Thus, with this understanding we decided on following *a hybrid modular-microservices based approach* for application design and development in our end-to-end IoT solution. This decision was also made keeping a future vision of the work in mind, where the microservices act as facilitators to enable dynamic service migration based on the network characteristics to increase quality of service and for better service provisioning.

The proposed approach needs to be incorporated at the very initial stages of the IoT application cycle, and it iteratively works through the successive stages. The required sub-steps in stage 1 and 2 of the IoT application life cycle of the proposed approach are presented in Fig. 3.4.

It is also important to note that the smart dairy farm setup in this work was limited to putting wearables on cows, and did not have any other sensors on the field, for e.g. soil monitoring or any other (arable farming etc.). With a diversity of sensors on the farm, and thus with more data, the set of services that can be provided may increase. Furthermore, there might be some limitations based on the sensors in use i.e., what kind of features can be generated or sensed by the sensors, and also what is the objective of the solution being

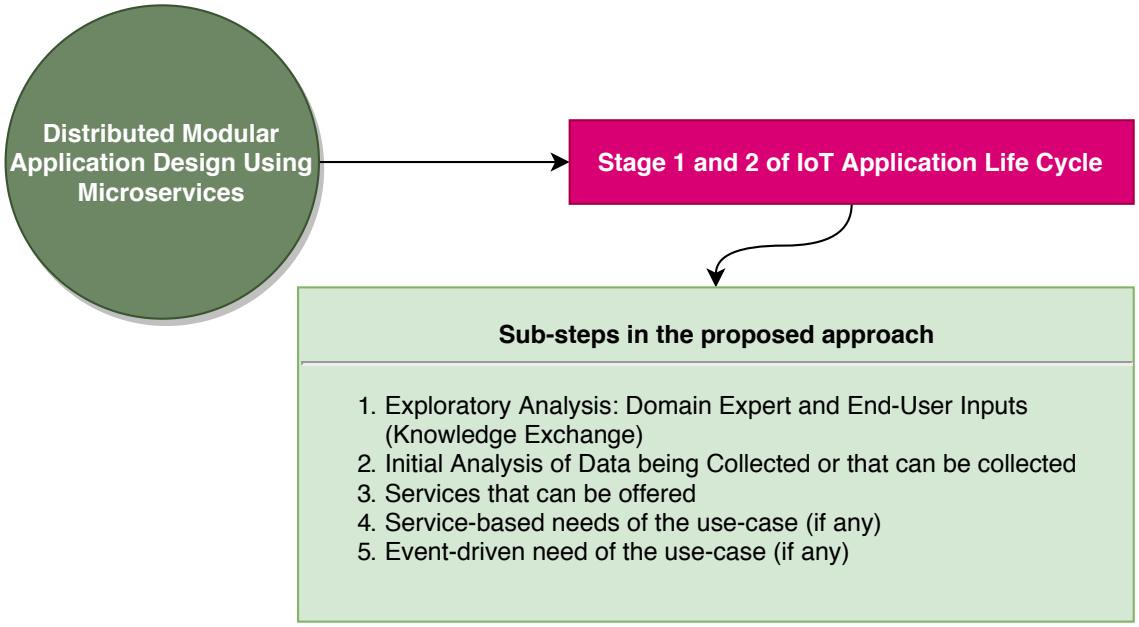


Fig. 3.4 The sub-steps of the proposed approach that need to be included in the stage 1 and 2 (plan and design) of the IoT application life cycle.

developed. For instance, in beef herds along with lameness detection, location tracking is also important in certain geographical regions such as South Africa to locate the herd for safety purpose and to ensure that they are away from wild animals such as mountain lions. In such scenario, it might be ideal to have the sensors that can provide the locomotion data and GPS coordinates of the cows. Thus, deciding on a vendor or more specifically the sensors to be used in the deployment primarily depends on the objective of the solution being developed.

In our scenario, the ultimate objective was early detection of lameness in dairy cattle, and lameness being dependent on locomotion scoring, it was ideal to use sensors that can provide locomotion data. We performed a market study to find out what other solutions use locomotion data and the associated dairy tech-service providers. We found that heat detection or estrus detection is a well-established market that uses the locomotion data to inform the farmers when an animal goes into heat and is ready to be inseminated for pregnancy. Thus, after reaching out to a number of service providers we went ahead with ENGS Systems, Israel as the finalized one to supply the sensors for our use-case for the **SmartHerd** project deployment in Ireland.

3.6 Microservices Oriented Architecture

The desired application has been built as a collection of microservices as shown in Fig. 3.5. We believe that microservices act as one of the key enablers to leverage fog computing while building an end-to-end IoT solution in such scenarios.

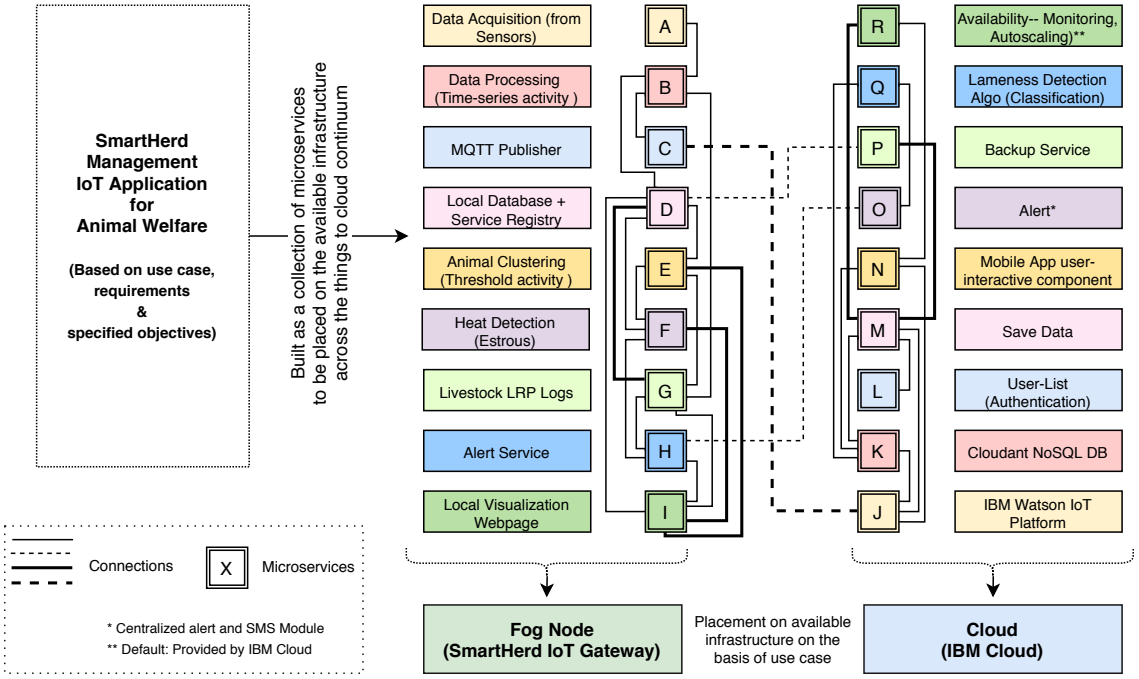


Fig. 3.5 Representation of developed SmartHerd System as collection of microservices and placement of these services on computational resources available in the infrastructure [7].

As mentioned before in Chapter 2 (section 2.7.1) that from an application deployment standpoint, fog computing can be perceived as component of application running at the network edge as well as in cloud, with the components running at fog being latency sensitive and time critical in nature. In our use-case scenario, the latency constraint and data requirement of a microservice remains static all the time. But this might not be the case with other use case scenarios — as an hypothetical example here, consider a microservice which takes temperature and humidity values from the sensors to determine environmental conditions, let's say in order to regulate air conditioning in an area. Now a sudden change in temperature might not mean anything, but if the change stays for a long time it might be an indication of fire in that area. Now in this scenario, the latency constraint and data requirement of the microservice will change. So, these two features of a microservice are use-case dependent.

In the case of detection of an event which would require attention towards the animal, an alert is sent out to the farmer. The end-to-end data and work flow of the developed application has been presented in Fig. 3.6.

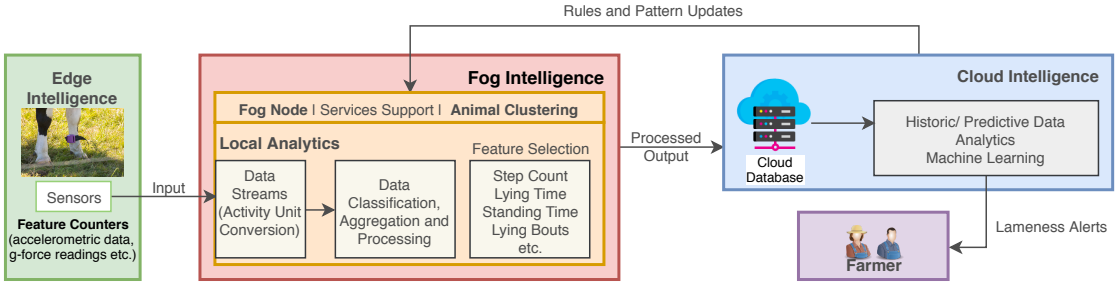


Fig. 3.6 Workflow and data flow in the testbed deployment [5], [7].

Orchestration and Choreography in the developed application: The individual service modules do not make any application workflow in isolation and need a method so they can interact, share data and provide the desired application workflow. This leads to the question of – *How to tie the service modules together for the desired application flow?* There are two ways to do it, that are used in practice — orchestration and choreography.

In orchestration, a single centralized entity usually referred to as the orchestrator coordinates the interaction among different services. It is responsible for invoking and combining the services. While in choreography, every microservice performs their actions independently. It does not require any instructions. It is like the decentralized way of broadcasting data known as events. The services which are interested in those events will use it and perform actions. The main difference between choreography and orchestration is in terms of where the logic that controls the interactions between the services resides.

The general norm is to have the developed services abstracted inside containers. The realization of service abstraction becomes more important in scenarios where fog devices are heterogeneous in nature and can range from end user devices to access points, routers to switches; so to accommodate such heterogeneity service abstraction is desired, and can be realized in terms of containerization. The examples of container technologies include Linux containers, Docker [171], etc. ; and from programming platform perspective Java Virtual Machines (JVM), Python Virtual Machine (PVM) can be used as an equivalent substitute. A wise decision on deciding which container technology to choose acts as an important factor in the efficiency and performance of the overall system. In our deployment setting, we used PVM and JVM as container technologies (*equivalent substitutes*) on fog node, and the default container technology in the IBM cloud (i.e., Docker as platform and Kubernetes as container orchestration system for Docker containers) while building the SmartHerd IoT system.

In our setup, we used orchestration for the service collaboration within fog node, and choreography for outside fog node and in cloud. Fig. 3.7 gives a representation of the same.

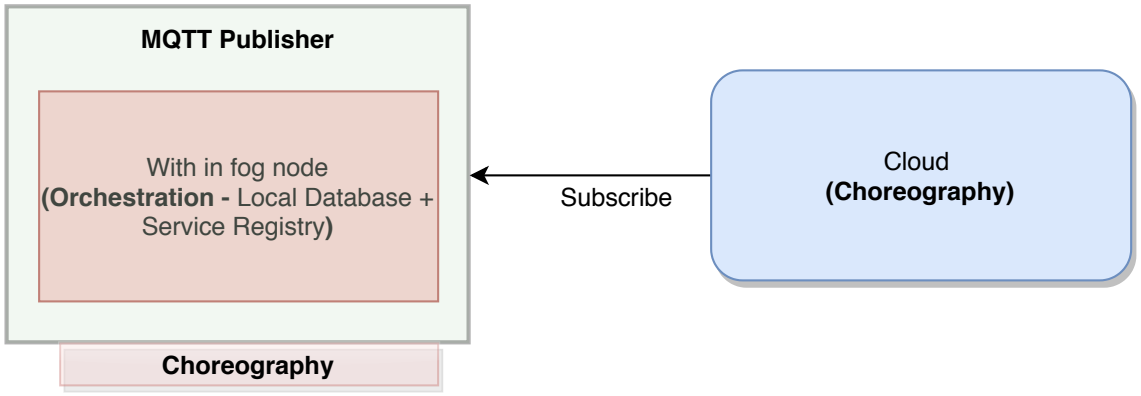


Fig. 3.7 Orchestration and Choreography implementation in the developed system.

We wanted to keep the service management overhead on fog node as low as possible. The programming platform (languages) used in the development process were Python and Java. Keeping track of developed modular part of the code base as isolated processes (Namespaces and Cgroups) gave us a way to use them as individual and different services with a simple service registry (in form of a simple file) rather than using an additional software/framework to do the same, and hence we used PVM and JVM as container equivalents on fog node.

The above is what contributed to the methodology being hybrid in nature i.e., modular-microservices approach. The designed and built microservices of SmartHerd IoT ecosystem are placed and run on the computational resources available in the infrastructure i.e., SmartHerd IoT Gateway (fog node) and cloud. The implementation is also hybrid from choreography and orchestration perspective.

3.6.1 Application Deployment in SmartHerd

The decision on which services to deploy on fog node and which on cloud node was made based on a number of factors presented below:

- Some restrictions arising from the sensor provider, such as that the data acquisition and data processing component necessarily have to be placed on fog node because of its involvement with the heat detection service.
- Correspondingly, heat detection service being a black box for us in the SmartHerd code-base had to be placed on fog node. This was part of the setup from the sensor provider for the heat detection offering to the farmer.
- The deployment decision for remaining services was based on data requirement analysis presented in table 3.1, and as per the application workflow. For e.g., heat detection is a latency sensitive service with immediate data requirement, and thus

should ideally be placed on fog node. Thus, as soon as the data is available, it can be used by the service without any additional transmission delays.

- Some of the services were bound to be deployed in cloud because of the connected association with IBM cloud for e.g. IBM Watson IoT Platform.

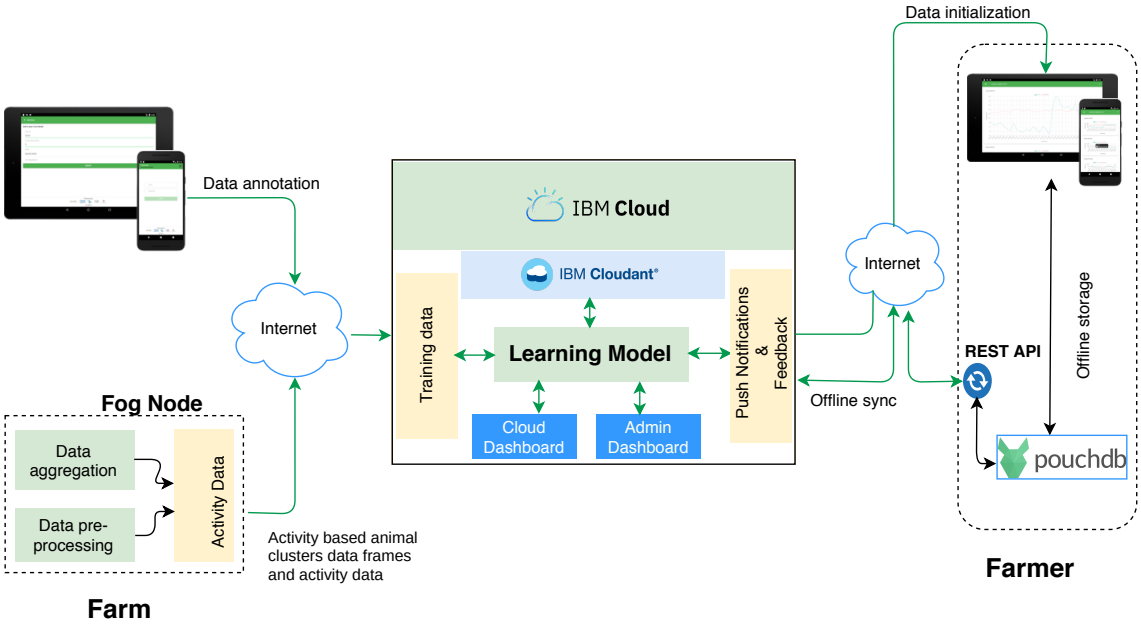


Fig. 3.8 Mobile application developed specifically considering the needs of the farmer, including an offline first strategy. The figure presents data and notification flow in the developed mobile application [9].

3.6.2 Offline-First Model for Mobile Application Design

We also developed a mobile application for the farmer to use and be notified of alerts from the developed system. The application works on both android and iOS operating system platforms, and was developed using Ionic Framework 3.0 [172] at that time, which is an open-source cross-platform mobile application development framework.

Farms are usually located in geographically remote locations facing constrained network connectivity. Most of the IoT deployments in such settings are faced with limited cellular coverage. Existing solutions are mostly cloud based or completely offline. This limits the farmers' ability to interact with the application anytime and anywhere. The system developed in this study uses an offline first strategy via the mobile application and cloud dashboard. Fig. 3.8 shows the data flow of the offline enabled design approach.

Once the model produces notifications, these are sent to the farmer's mobile device as push notifications. On board the application is a PouchDB [173] database which

synchronizes with the cloudant database in IBM cloud using a REST API whenever a connection is established. The application in general helps to achieve the following tasks:

- Push notifications: Whether on WiFi or limited cellular network, or whether the application is open or not, these will go through each time the status of the farm changes.
- Data annotation: During the training process, this feature was used by the human operator to annotate the data. In our case, this was done weekly by an agriculture student.
- Feedback to improve model learning: When a notification is generated, the farmer has the option of confirming if the said cow is actually lame, or tag it as a false alarm or even report a missed alert. All this information is sent back to the model to improve its accuracy.

3.6.3 Multi-Vendor Interoperability of the Developed System

As mentioned earlier, that unlike the existing systems that are based on a monolithic design approach, the application designed in this study follows a microservices based approach for design, creation and deployment. The aim was to make the developed system as 'Application/Software as a Service', which can be used by the service providers to integrate with their existing systems. For example, an agri-tech company could be a service provider for any other solution such as mastitis detection, who wants to expand their system or integrate any of the services such as lameness or heat detection into their system. A visual representation of such a possible integration is presented in the Fig. 3.9. Feature engineering layer as shown in the Fig. 3.9 ensures that data is transformed to output only the required features and also reject those that cannot be engineered to form the required features for a desired service: For example, Lameness Detection and Heat Detection Service expects lying time, step count and swaps, but a service provider might have activity counter instead of step count, and Stand up + Liedown instead of swaps.

It is important to note that this layer will be different for each service provider, since the underlying sensor technology might be different. This is turn makes the developed system sensor agnostic. The output from feature engineering layer is then passed to the access layer, which includes both mobile and web components. This then goes through a REST API which in turn calls the desired service.

Vendor software stack interoperability: In the SmartHerd deployment, the heat detection service provided by ENGS system was integrated into our developed software stack. This interoperability was possible because of the microservices oriented design approach. The interoperability feature is bi-directional, but it was easier for us to integrate

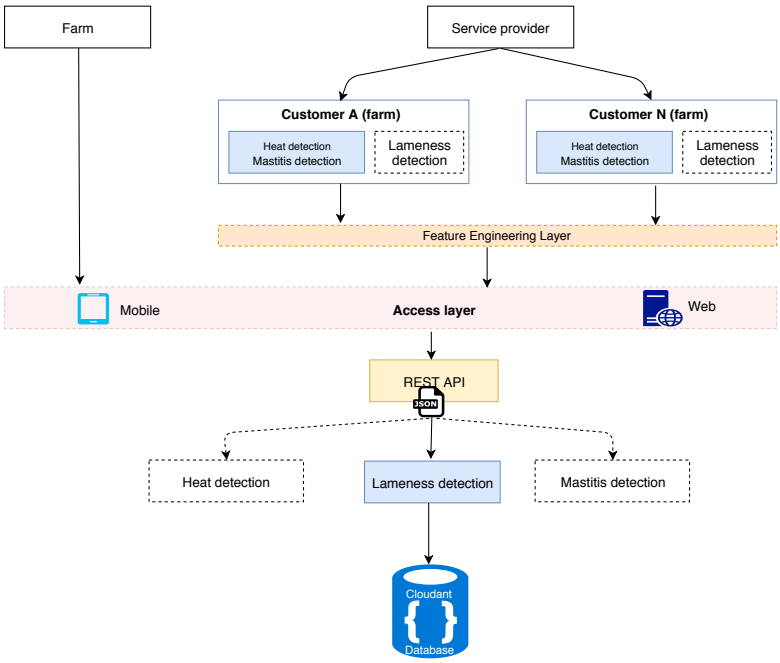


Fig. 3.9 Proposed microservices based application design and flow for integration of services from different service providers [6], [9].

their heat detection service in our software stack as accessing their code-base for the integration required additional legal approval from the involved entities.

So with their approval we incorporated their heat detection service to demonstrate interoperability within the SmartHerd deployment. Although it should be noted that the developed system (and approach) is multi-vendor interoperable as well, which is being validated as a part of the ongoing and future work in the MELD project with two vendors as sensor technology providers.

3.7 Results and Discussion

A short demo-video of the developed system is available at [174]. The research output presented in this chapter was collaborative work with colleagues within our organization, and a mentor from industry partner IBM, Ireland.

As the overall work has two objectives — one from fog computing perspective, and other from machine learning perspective, thus there were contributions made into various places and stages of the project. An illustrative representation of contributions made as **major** and **minor** in the project has been shown in Fig. 3.10.

For a better representation and analysis, the results from this work have been divided into three parts as below:

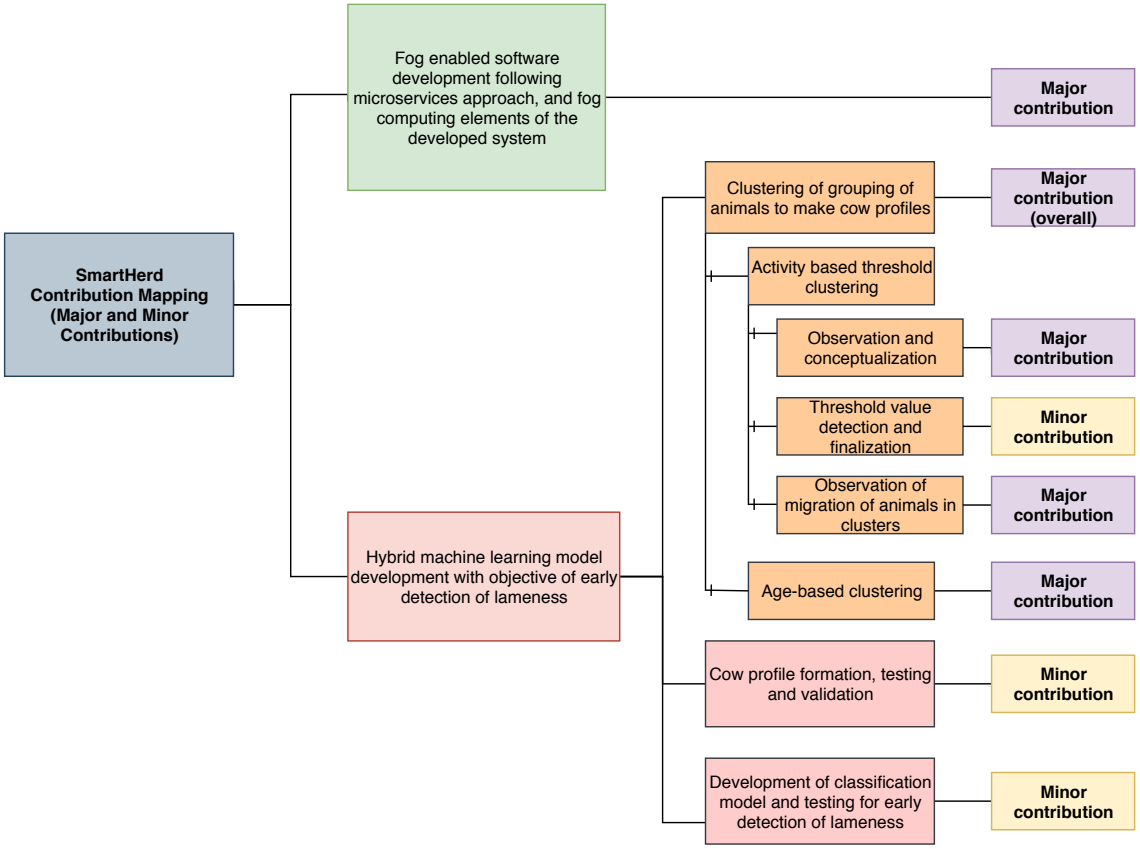


Fig. 3.10 Diagrammatic representation of the SmartHerd work mapped to contribution share made in the project.

(1) Fog computing functionality and objective of the developed system (sub-section 3.7.1)

(2) Analysis of application design and development approach coupled with the fog computing paradigm (sub-section 3.7.2)

(3) Machine learning objective of the developed system (sub-section 3.7.3)

Fig. 3.11 presents a graphical mapping of the results, which have been presented in detail in the sub-sections below.

Some specifications and terminologies used in the sections 3.7.1, 3.7.2 and 3.7.3

There are few terms that have been used to describe the results presented in sections below. The details on these terminologies and specifications have been presented here:

1. For a comparative analysis, we have compared the fog functionality of the system in two experimental scenarios. First one, is **with fog functionality** and is termed as

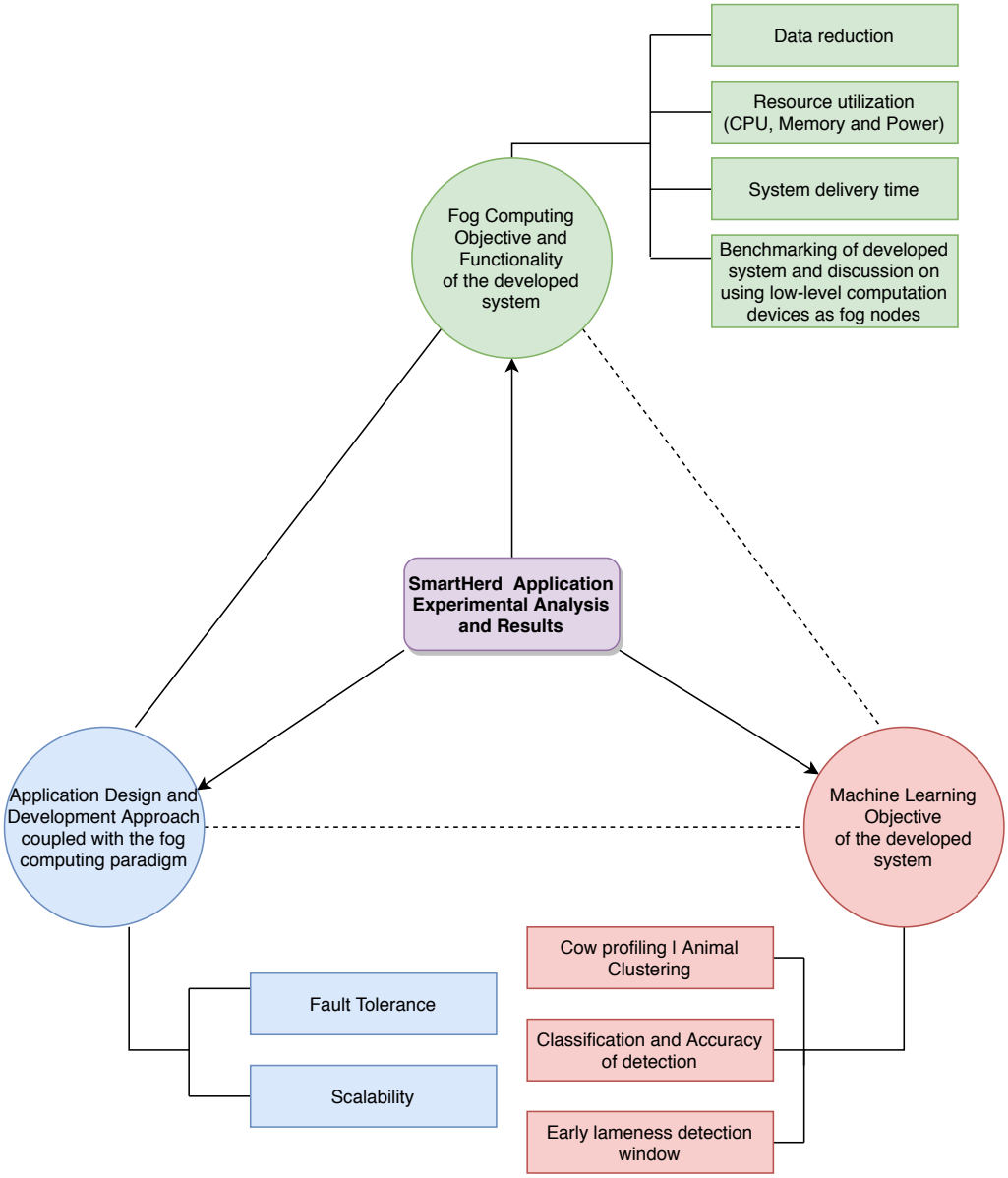


Fig. 3.11 A diagrammatic representation of the analysis presented for the solution developed in SmartHerd setup. It represents the SmartHerd work mapped into fog computing objective, application development approach and machine learning objective of the developed system. The solid line in the triangular representation indicates that the analysis presented is a combination of two vertices. The dotted line indicates vertices being a part of the developed solution, but there is no direct association between them for the experimental analysis presented, other than being a part of the whole end-to-end solution developed.

fog assisted approach. Over here the fog node has all the services deployed on it as presented in Fig. 3.5. Second is **without fog functionality** and this is termed as **cloud-centric approach**. Over here, the fog node only serves as a usual gateway to

transport the collected data to the cloud node without performing any computation on it.

2. In both scenarios i.e., with fog functionality enabled (fog assisted approach) and without fog functionality enabled (cloud-centric approach), the response to the user is sent from service running on cloud node. The user, which in the SmartHerd project deployment is the farmer, interacts with the system via the mobile application. All the generated alerts are sent via the mobile application, which also serves as the medium through which the user can query the system. The user has also been provided with a functionality to mark the alert or queried output presented from the lameness detection service as either correctly classified or wrongly classified.
3. The data streaming frequency of sensors to fog nodes is every 6 minutes, and is specified by the sensor-provider in the deployment. The data streaming frequency of the fog node to transmit processed data to the cloud node was set to an hourly basis.
4. The network available on user's mobile phone was set to 3G for all the experiments.
5. For development of the lameness detection algorithm, in the initial stages of the project the analysis was done on the hourly basis; but this was later switched to a daily analysis of the data as that was more insightful for the early lameness detection objective. All the feature and data analysis graphs presented in sub-section 3.7.3 have been presented with the daily analysis of the system. Thus, the development of lameness detection algorithm was performed on daily summarized data, and the graphs presented in that section are for each activity on a per day basis.
6. We started with 150 cows each having a pedometer on one of their front legs in the deployment. During the life span of the project, a total of four cows were eliminated from the analysis for miscellaneous reasons. Thus, the total number of animals in the project varied from 150 (at the beginning of the project), to 147 (at mid-stage of the project), to 146 (at final finishing stage of the project).
7. Table 3.2 presents the resource specification of the fog node and cloud node, and other associated context in the SmartHerd project test-bed setup.
8. The tool used to monitor CPU and Memory consumption was psutil [175] python library. It is a cross-platform library for process and system monitoring in python.
9. Other than having a backup service on cloud node, there was no additional functionality added to the developed system to increase the fault tolerance or system resilience.
10. **System access by the user — User-Access Queries for the system:** As mentioned before, the user in the current setup of SmartHerd is the farmer in the loop. A user can make a query to the system via the mobile application. The possible queries can

Table 3.2 Resource specification of fog node and cloud node, and other associated context in the SmartHerd deployment.

Number of animals on the farm, which corresponds to the number of sensors on the farm	150
Number of fog node (s) and it's resource configuration	1 Configuration: Intel® Core™ 3rd Generation i7-3540M CPU @3.00GHz, 2 Core(s), 4 Logical Processor(s) 16.0 GB RAM, 500 GB Local Storage
Number of cloud node (s) and it's resource configuration	1 Configuration: 8 VCPUs Intel® Core™ Xeon E3-1270 v5 4 Cores, @3.60 GHz 16.0 GB RAM, 500GB Hard disk
Data transmission frequency of sensors to fog node	Every 6 minutes
Data transmission frequency of fog node to cloud	Hourly basis

be divided into user-triggered queries and system-triggered queries. A list of some of the possible queries have been listed below:

Query 1: The user chooses a random cow to view the locomotion pattern and historical lameness report. For this we select a random cow from the available list and make this query, and we take a note of the time taken by the system to respond to this.

Query 2: The user chooses a random cow to check whether it is lame or not. Over here we select a random cow from the list and take a note of two things: 1) the time taken by the system to respond to the query 2) the farmer's input– either confirmation or rejection of the classification output presented by the system. The user's annotated response as correctly classified or mis-classified is useful in further retraining the machine learning element of the developed solution to improve accuracy.

Query 3: The system detected an animal (or a list of animals) as lame, and the user needs to be informed of those. For this the same metrics are noted down as in Query 2. The difference between Query 2 and Query 3 is that Query 3 is system triggered while Query 2 is user-triggered.

Query 4: The user picks a random cow and wants to check whether it's in heat or not.

Query 5: The estrus detection service detected that an animal (or a list of animals) are in heat, and the user needs to be alerted of them.

For Query 4 and Query 5, we treat the Heat detection service provided by ENGS System as a black box. As stated earlier, the main aim of integrating the heat detection

service in our software stack was to show the interoperability feature of the developed system. Therefore for us, it acts as a black-box to generate the output on a user-triggered query or send the output in case of system-triggered events.
Please note that the queries listed above are transformed as standard NoSQL queries from a code-base standpoint.

11. **Service Delivery Time:** Service delivery time refers to the time when the query was made, plus the processing time by the system, plus the time it took for the response to reach the user. It is therefore the sum total of overall transmission and computational time.

We limit our selves to user-triggered queries of type 1 and 2 (i.e., Query 1 and Query 2) stated above in the experimental analysis of service delivery time and fault tolerance. In the service delivery time experimentation, each query was repeated 30 times to collect the metric. The experimental results on service delivery time have been presented in section 3.7.1.3. For the fault tolerance experiment, each query was repeated 3 times to note down the response time. The reason behind the difference in the number of times the queries are being repeated for collecting the metric is that in-case of a normal service delivery time experimentation, the primary objective is to see the average response time of the system to the user. While in case of a fault tolerance experiment, as the system is already getting disrupted, the aim is to check the working of system in such disruptions. The large sample size of values in first case is useful to analyze the average response time with a confidence interval, while in second experimentation the objective is to see how the system works with services getting broken randomly i.e., to check the fault tolerance of the developed system.

3.7.1 Fog Computing Functionality of the Developed System

3.7.1.1 Data reduction

Among the downsides of the existing approaches is that they are either fully cloud-centric in nature, i.e., all the data is sent to the cloud for processing and analysis; or have just farm premises based system which limits the accuracy and intelligence [114] of such systems as there are no dynamic and frequent updates.

From what we have monitored, each sensor/LRP generates around 70 KB of data per day, which, as shown in Fig. 3.12, accounts for 10.1 MB of data collected from the 147 sensors at the SmartHerd gateway each day, with data acquisition and streaming happening every hour. But given the local processing capability attributed to fog computing, the amount of data that is sent further to the cloud is only 1.62 MB per day, which, as shown in Fig. 3.12, is an 84% reduction in the amount of data that would otherwise have streamed to the cloud throughout the day. This aspect of data reduction becomes even more crucial

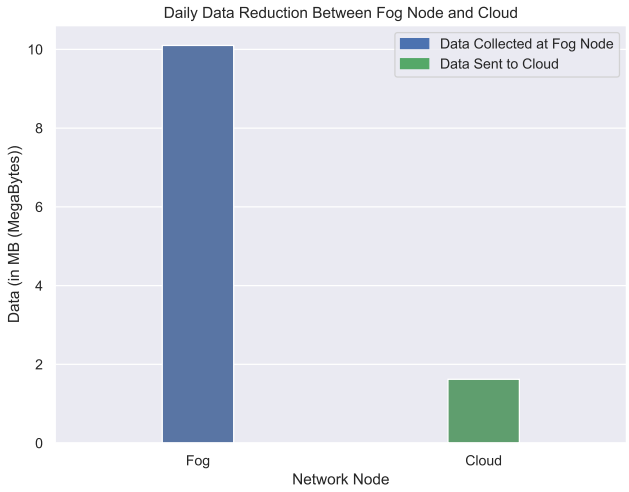
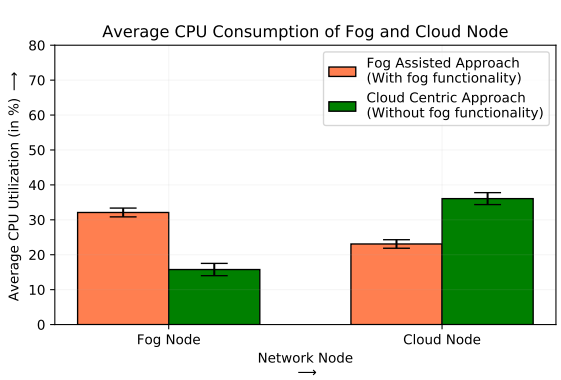


Fig. 3.12 Daily reduction in the amount of data between the fog node and the cloud.

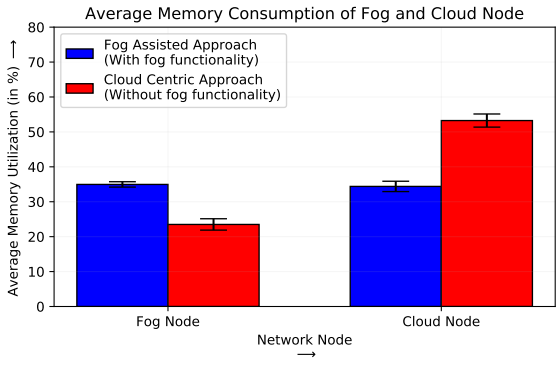
while scaling up the farm and the herd, as the amount of data collected and streamed would rapidly increase.

3.7.1.2 CPU and Memory utilization with and without fog functionality

The bar plots in Fig. 3.13 presents the CPU and Memory utilization collected using psutil with a time step of 3 minutes window as the utility parameter. It was captured for a period of 12 hours, and averaged over the entire period for a better representation. The whiskers present the 95% confidence interval for the values presented.



(a) Average CPU utilization at fog and cloud node



(b) Average Memory utilization at fog and cloud node

Fig. 3.13 Average CPU and Memory consumption of fog and cloud node with and without fog functionality.

The CPU and Memory utilization of fog node is more in fog assisted approach and less in cloud centric approach as visible from the bar plot presented in Fig. 3.13a and 3.13b. For cloud node, the CPU and Memory utilization is more in cloud centric approach as compared to fog assisted approach. This is because of the fact that in fog assisted approach, there is additional computation that happens on the fog node which results in an increase in computing resource utilization; while in cloud centric approach, the fog node only serves as a gateway to forward the data to cloud node for processing.

The same goes for cloud node as well, in cloud centric approach, every computing operation on the data received happens there. While in fog assisted approach, the cloud node receives processed data, wherein a part of the computing operation is already done by the fog node, which results in less computing resource utilization in fog assisted and more in cloud centric approach for cloud node.

3.7.1.3 Service Delivery Time experimentation with and without fog functionality

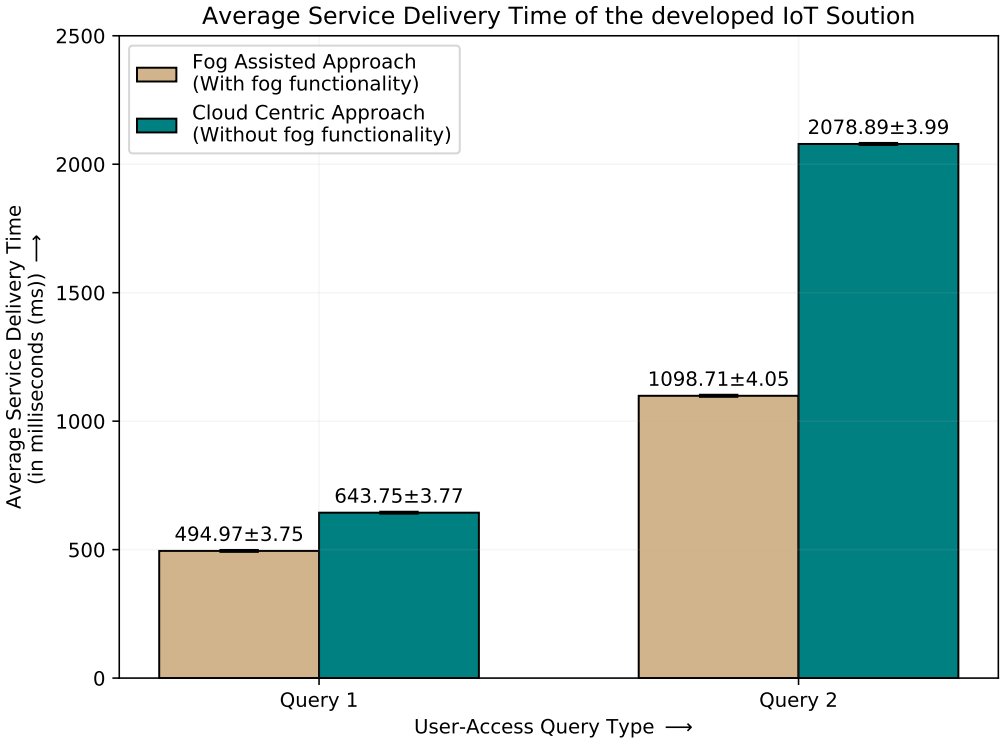


Fig. 3.14 Service Delivery Time of the developed SmartHerd IoT solution with and without fog functionality.

The service delivery time here represents the average query time that we performed on the system with different queries as defined earlier in 3.7.2 for user-triggered Query of

type 1 and 2. It includes the network round trip time along with the processing delays by the system.

Fig. 3.14 presents the service delivery time of the developed system. The value listed on the top of the each bar presents the service delivery time with number after the $\pm$ symbol denoting the 95% confidence interval for the value. As visible from the plot, the service delivery time is less in fog assisted approach and more in cloud centric approach for both Query 1 and Query 2.

With fog assisted approach it is less, as the animals here are already clustered (i.e., part of computing operation has already been performed by the fog node) and the user-access Query directly executes the animal as a data-query point. While in cloud centric approach, the animal being queried has to be put into a cluster first and then treated as a data-query point to send the output. Thus, the additional time in cloud centric approach corresponds to that. The animal clustering process used in the setup has been explained in detail later in section 3.7.3.2.

3.7.1.4 Benchmarking of the developed system, and discussion on platform performance on using fog node with low-level computational power

As the real world test-bed had a high-end fog node (computer), there were a number of obvious questions listed below that were a required ask for a thorough analysis of the developed system:

† How does the designed software system perform when a fog node with lower computation power is set as a gateway?

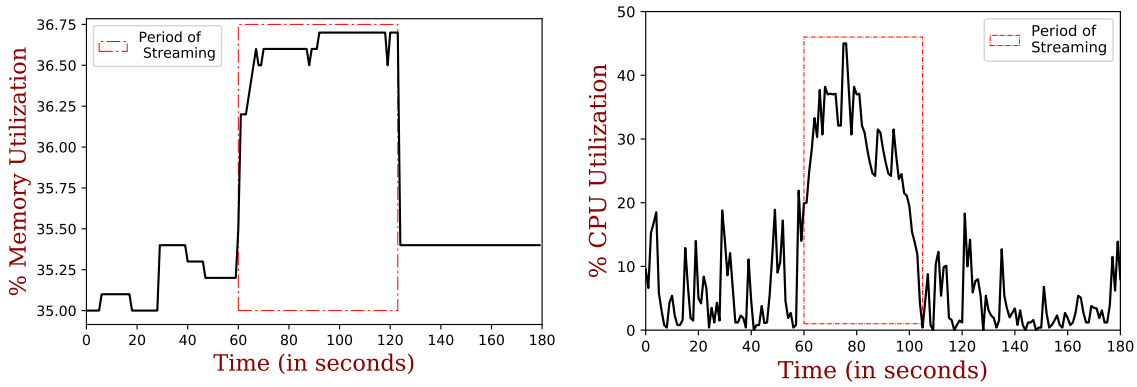
† Is the developed system capable of running on resource constrained devices having lower computation capacity?

† What are the minimum resource requirements for the fog node in such a setup?

These questions lead to benchmarking of the developed system. As per the application deployment made, we are aware that the main processing component amongst the microservices running on fog node is the MQTT publisher which is responsible for data pre-processing, aggregation, and streaming processed data to the cloud. For initial analysis ultimately leading to benchmarking of the system, we first zoom into the CPU and memory analysis data captured for analysis in section 3.7.1.2. i.e., we monitored resource (CPU and Memory) consumption on fog node every 3 minutes when all the microservices were running on it. Using psutil python library, we calculated the percentage increase in both CPU and Memory before, during and after streaming (i.e., before the MQTT publisher was run, during and after it was run). We used the percentage increase because this would form

a baseline here i.e., irrespective of change in the fog node capabilities, the same increase in resource consumption would be expected.

Such an analysis will help us in designing the experimental setup for benchmarking the system to find the minimum computing resource requirement. The idea is to check the postulate made above, which states that irrespective of change in the fog node capabilities, the same percentage increase in resource consumption would be expected.



(a) Memory utilization at fog node

(b) CPU utilization at fog node

Fig. 3.15 Resource utilization at fog node (i.e., SmartHerd IoT Gateway).

Fig. 3.15a shows memory usage before, during and after the MQTT publisher is run. The highlighted region shows the period during which the MQTT publisher was streaming. As it can be seen, there is an increase from about 35.38% to 36.75%. The total usable memory was about 14.50 GB. This would mean the impact of MQTT component was only 0.12 GB.

Fig. 3.15b, shows CPU usage before, during and after the MQTT publisher is run. The region highlighted shows the period during which the MQTT publisher was streaming. As it can be seen, there is an increase from ≈20% to ≈45%.

The table 3.3 summarizes the CPU and Memory utilization before, during and after streaming.

In all, the overall resource utilization and effects of MQTT component on the fog node are very minimal validating that it can be run on very resource constrained compute environments.

It should also be mentioned here that although the real world experiment was carried on a high-end fog node (computer), but prior to this experiment (i.e., before putting the developed code-base in the actual deployment), we tested the resource (CPU and Memory) consumption on another PC setting (Intel® Core™ i5-4300M CPU@ 2.60GHz, 8.0 GB RAM, 500 GB local storage), and also on a single VM instance running on OpenStack with following configuration: 2 VCPUs (Intel® Xeon Processors @2.60 GHz), 4 GB RAM,

Table 3.3 CPU and Memory utilization on fog node before, during and after streaming for comparison of the developed system to be able to run in resource constrained environment as well.

Resources	Before streaming with all the microservices running	During streaming with all the microservices running	After streaming with all the microservices running
CPU	At max 20% and mostly below that	From 20% to 45 % (25% increase)	At max 20% and mostly below that
Memory	At max 35.38% and mostly below that	From 35.38% to 36.75% (1.37% increase)	At max 35.38% and mostly below that

20 GB Disk, initially with artificial data and later with data obtained from ENGS systems (our sensor vendor for this deployment). Similar behaviour as per the aforementioned postulate was observed there as well.

Power Consumption:

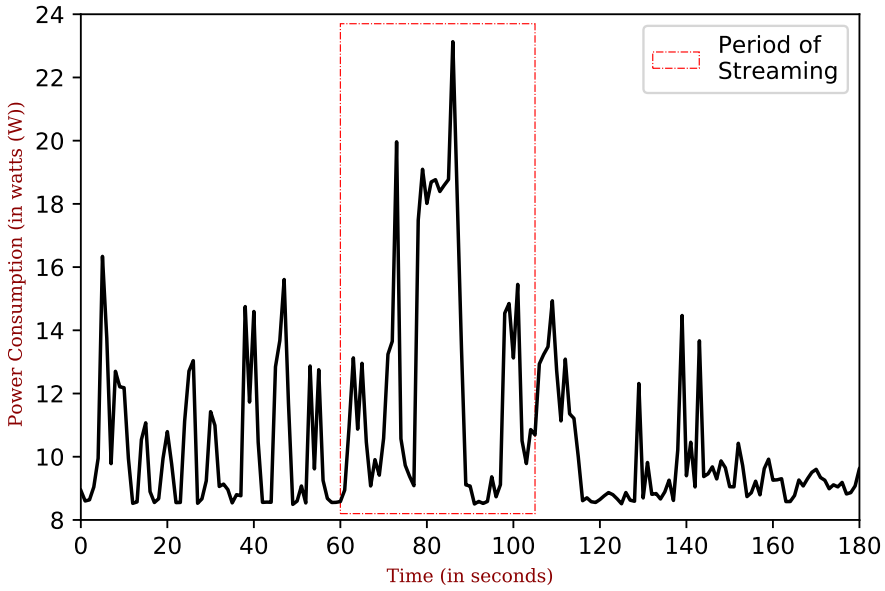


Fig. 3.16 Power utilization at fog node.

We also captured the power consumed at fog node as that would also help in the analysis of benchmarking of the developed system. We used Intel® Power Gadget tool [176] to measure the power consumption of the fog node (i.e., SmartHerd IoT Gateway). It is a software-based power usage monitoring tool enabled for Intel® Core™ processors. While we collected the data for a longer period during the deployment for this analysis, we observed a constant power consumption pattern throughout. Thus, to give a better representation of the power consumption pattern, we present the power consumption at the fog node in a time window of 3 minutes in Fig. 3.16. As visible from the plot, the

maximum utilization is 23.2 Watts, and for most of the time, it stays below 18 Watts. The utilization reaches its maximum value (during the streaming period) when there is a message stream happening between fog node and cloud node.

Benchmarking of the developed system – experiment and analysis

It should also be mentioned here that the computing resource benchmarking in such a setup is primarily concerned with the fog node in the setup i.e., analyzing the effect of varying computing resources at the fog node while keeping the resources on cloud constant. The justification for this is that at fog node auto-scaling (adding and removing computing resources dynamically) of resources is not possible, unlike cloud computing environments.

Without affecting the integrity of the experiment and analysis here, it can be assumed that the cloud node is capable of managing the computing resource for such as setup dynamically while fog node currently is not. There is no doubt that there are research efforts in that direction for fog computing environments as well, but the dynamic resource allocation to services in fog computing environments is not in the scope of the work presented in the dissertation.

The benchmarking of the system was done to check the minimum amount of computing resources required to run the developed solution in the SmartHerd deployment. The experimental testing here is done with one user and application deployment as presented in Fig. 3.5. The idea was to check with such a setup where there are 150 animals with wearables on them generating data for the desired analysis, what is the minimum computing resource capacity required for fog node. Table 3.4 presents further insight into the benchmarking experimentation. Please note that CPU and Memory utilization is noted down every 3 minute window period as the time step for a total of 3-time windows in the 15 minutes duration of each set of experiment separately (i.e., each row here). In those three time windows the data streaming from fog node to cloud node happens only once because of hourly streaming frequency at fog node. In these set of experiments, the streaming happened in the second time window.

Table 3.4 Benchmarking Experimental Scenario — Keeping computing resource on cloud same and varying resources on fog node.

Limiting Allocated CPU (Varying Physical CPU (1 to 4))	**Limiting Allocated Memory Varying Memory (2GB to 16GB)**	**Metrics Noted Down**
1	2	Note down CPU and Memory Utilization of allocated resources, and attribute them as follows: 1. Average CPU and Memory Utilization during the experiment 2. CPU and Memory utilization before, during and after streaming 3. % increase in CPU and Memory utilization during streaming
2	4	
3	8	
4	16	

Methodology used for benchmarking — limiting allocated computing resources with Cgroups: The full form of Cgroups [177] is Control Groups. It is a Linux kernel feature used to limit resources to process groups such as CPU, memory, permissions and many more on Linux. We use Cgroups to perform benchmarking of the developed system. We varied the available computing resource capacity on fog node using Cgroups as listed in Table 3.4.

Further details on the use of Cgroups has been included in appendix B. The output obtained from the benchmarking experimentation have been presented Table 3.5 and 3.6.

Table 3.5 CPU utilization of fog node during benchmarking experiment.

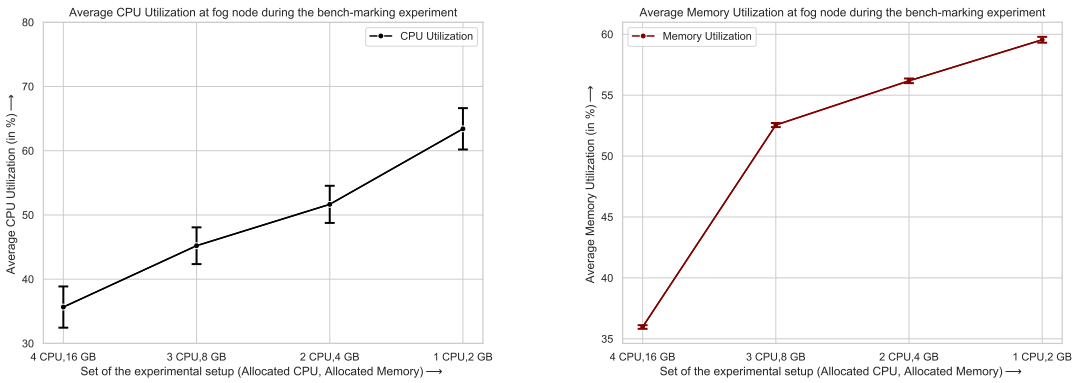
Allocated CPU	**Allocated Memory**	**CPU utilization of fog node**				
		Before streaming	Increase in value during streaming	After streaming	% increase value during streaming	Average value for the duration (Value ± 95% confidence Interval)
1	2	≤50.72	50.72 to 76.53	≤50.72	25.8	63.41 ± 3.22
2	4	≤39.26	39.26 to 64.96	≤39.26	25.7	51.65 ± 2.89
3	8	≤31.84	31.84 to 57.34	≤31.84	25.5	45.21 ± 2.86
4	16	≤20.12	20.12 to 45.23	≤20.12	25.1	35.66 ± 3.21

Table 3.6 Memory utilization of fog node during benchmarking experiment.

Allocated CPU	**Allocated Memory**	**Memory utilization of fog node**				
		Before streaming	Increase in value during streaming	After streaming	% increase value during streaming	Average value for the duration (Value ± 95% confidence Interval)
1	2	≤58.82	58.82 to 60.66	≤58.82	1.84	59.55±0.24
2	4	≤55.27	55.27 to 56.93	≤55.27	1.66	56.18±0.19
3	8	≤51.75	51.75 to 53.27	≤51.75	1.52	52.55±0.17
4	16	≤35.38	35.38 to 36.75	≤35.38	1.37	35.97±0.15

As expected that with decreasing amount of computing resources on fog node, the CPU and memory utilization of fog node increases, as also visible from the plots presented in Fig. 3.17. The whiskers in plots present the 95% confidence interval for the values obtained.

Also, as analyzed from the % increase column in Table 3.5 and 3.6, the increase in resource utilization is not with the exact same value, but it still appears to be in a very close range of that, which appears to be almost the same value as postulated above. Moreover, it



(a) Plot representing increase in average CPU utilization at fog node with decreasing quantity of allocated CPU.

(b) Plot representing increase in average Memory utilization at fog node with decreasing amount of allocated Memory.

Fig. 3.17 Benchmarking experiment analysis — CPU and Memory utilization at fog node.

still supports that the developed system is fully capable of running on fog devices with low computational capacity.

We also agree that the computing resource benchmarking can further be abstracted to number of CPU cycles required to run the computing operations on the machine, with a similar analogy for Memory as well. We did look for options that can help us achieve that, but could not find something constructive and finally selected using cgroups for this, and presented the results obtained as per the granularity of the underlying methodology. This is one of the possible future directions for the work for further abstracting the computational resource benchmarking of such fog enabled IoT solutions, and has been mentioned in Chapter 6.

Our objective was to see the effect of code-base deployed on fog node in our setup while varying the amount of allocated computing resources. Such an analysis becomes essential to consider for possible scenarios where a single fog node hosting multiple services from different providers is in use, such as in a more heterogeneous IoT deployment scenarios. The minimum resource requirement of fog node in current setup from the analysis presented above is 1 physical CPU and 2 GB of Memory.

3.7.2 Analysis of Application Design and Development Approach Coupled with the Fog Computing Paradigm

As the distributed modular application architecture is using microservices based software development approach in SmartHerd deployment; thus, for experimental analysis, the objective was to analyse how a fog assisted approach helps the developed application.

A standalone analysis of application development approach without considering the fog computing paradigm would only have shown the benefits arising from using a microservices based software development methodology. Fig. 3.11 gives a presentation of the combinations of different aspects and objective of the SmartHerd IoT solution as presented in the analysis.

The novel contributions coming from this part of the work can be summarized as below:

- To the best of our knowledge and literature survey done, microservices based software development methodology has been limited to the cloud computing environments. It has only recently gained attraction to be used in fog computing environments, and there exists very limited work on using it in fog enabled IoT scenarios, particularly in a real-world setting.
- The novelty of the work presented comes from the point of applying the distributed modular application architecture using microservices based software development methodology coupled with the fog computing approach in the developed IoT application.

We used fault tolerance and scalability as the metrics to measure the effectiveness of the proposed application design and developed approach coupled with the fog computing paradigm.

Fault Tolerance: One of the key motivations for having a distributed system design is fault tolerance. Fault tolerance implies that the system still works well even when some parts of it are broken.

From a software standpoint and for the experimentation purpose, fault tolerance was implemented by means of timeouts, exception handling, and incremental restarting of the service. The idea was to break some part of the services and see the response of the system in such events.

We used a stochastic methodology to decide which services to break based on a probability function. The main aim of this was to keep the nature of the experiment random, i.e., to not selectively decide which services are working and which are not. Table 3.7 presents a description of the probabilistic approach used for disrupting the service randomly in the developed software solution for the purpose of experimentation. It should also be mentioned here that it was not possible to use a tool such as Chaos Monkey [178] for doing fault tolerance experimentation because of security restrictions imposed by IBM Cloud. Thus, we had to write our own controller code with the stochastic methodology and software standpoint described above for the purpose of experimentation with admin privilege access to all the services.

Table 3.7 Experimental settings for the fault tolerance experiment for randomized disruption of services in the developed solution.

Probability Value	Interpretation
0	Probability value of 0 means that none of the services are broken.
0.2	Probability value of 0.2 means that 20% of the total available services are broken
0.4	Probability value of 0.4 means that 40% of the total available services are broken
0.6	Probability value of 0.6 means that 60% of the total available services are broken
0.8	Probability value of 0.8 means that 80% of the total available services are broken
1.0	Probability value of 1.0 means that 100% of the total available services are broken

The maximum wait time to get a response for query was kept as 10 seconds, and if the user does not get a response by then, then an error message is displayed to the user. It should be noted that it was essential to setup a max limit in-case if there was no response from the service in case of it being broken as part of the fault tolerance experiment. The wait time of 10 seconds before sending an error message was kept in line with the usual response time limit of users experience [179] and also inline with the usual response time of the developed system for these queries without any disruption as presented in section 3.7.1.3. If the called service is not available, and if backup is available, the request is redirected to backup service and the response goes from there. For each service in case of unavailability, the wait time to get a response was kept as 200 ms before going and looking for backup service and serving the request from there.

As mentioned earlier, that both the Queries (Query 1 and Query 2) were run thrice. We performed the fault tolerance experiment in both scenarios i.e., with fog functionality (fog assisted approach) and without fog functionality (cloud-centric approach). The metrics noted for service delivery time in both scenarios for each execution of the experiment have been presented in Table 3.8 and 3.9.

In these tables, a '✓' means that the corresponding Query was served, a '✓B' means that the corresponding Query was served from backup service, and '✗' means that the Query was never served and the user received an error message. After carefully observing and analyzing Table 3.8 and Table 3.9, we present Table 3.10 which presents the general pattern observed by both Query 1 and Query 2 user access requests with and without fog functionality.

For a visual analysis of the results, we present two sub-figures in Fig. 3.18. Fig. 3.18a presents the number of the times the user access query was served in three experimental runs of the experimentation, and Fig. 3.18b presents the average service delivery time of those three experimental runs.

It should be noted that there were no additional measures taken to increase the fault tolerance of the system, and as mentioned earlier, the idea was to check the merits or

Table 3.8 Fault Tolerance experiment results with fog functionality i.e., in fog assisted approach.

Service Delivery Time in milliseconds (ms) is noted down in each experimental run for each Query, with 10 seconds (10000 ms) as the maximum wait time before sending an error message.							
Probability Value	**% of total services getting broken**	**1st Experimental Run**		**2nd Experimental Run**		**3rd Experimental Run**	
		Query 1	Query 2	Query 1	Query 2	Query 1	Query 2
0	0	495.87 (✓)	1096.89 (✓)	494.05 (✓)	1100.12 (✓)	495.04 (✓)	1097.62 (✓)
0.2	20	10000 (✗)	10000 (✗)	2498.97 (✓B)	3200.18 (✓B)	2502.06 (✓B)	3102.51 (✓B)
0.4	40	491.75 (✓)	1094.05 (✓)	2490.89 (✓B)	3108.35 (✓B)	493.28 (✓)	1101.71 (✓)
0.6	60	2497.67 (✓B)	3098.70 (✓B)	10000 (✗)	10000 (✗)	2490.91 (✓B)	3099.17 (✓B)
0.8	80	2500.18 (✓B)	3572.38 (✓B)	10000 (✗)	10000 (✗)	10000 (✗)	10000 (✗)
1.0	100	10000 (✗)	10000 (✗)	10000 (✗)	10000 (✗)	10000 (✗)	10000 (✗)

Table 3.9 Fault Tolerance experiment results without fog functionality i.e., in cloud-centric approach.

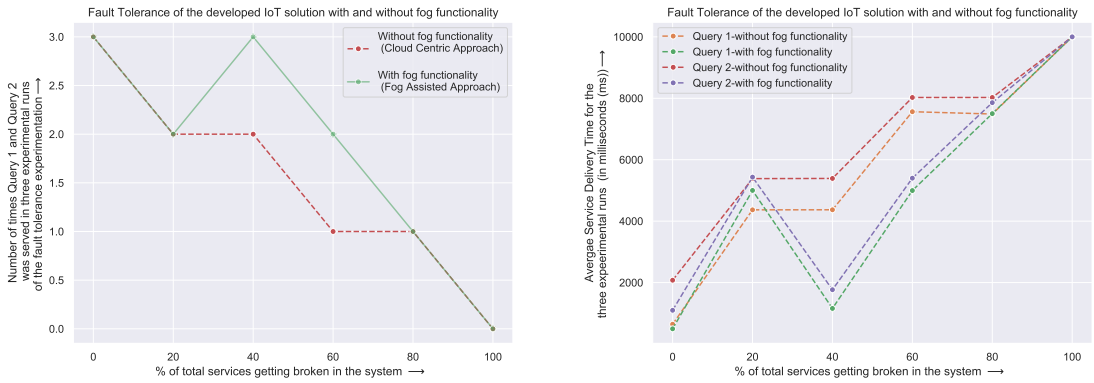
Service Delivery Time in milliseconds (ms) is noted down in each experimental run for each Query, with 10 seconds (10000 ms) as the maximum wait time before sending an error message.							
Probability Value	**% of total services getting broken**	**1st Experimental Run**		**2nd Experimental Run**		**3rd Experimental Run**	
		Query 1	Query 2	Query 1	Query 2	Query 1	Query 2
0	0	644.26 (✓)	2077.38 (✓)	642.35 (✓)	2076.96 (✓)	646.18 (✓)	2077.63 (✓)
0.2	20	2643.75 (✓B)	4078.89 (✓B)	10000 (✗)	10000 (✗)	640.77 (✓)	2075.32 (✓)
0.4	40	10000 (✗)	10000 (✗)	2465.26 (✓B)	4089.32 (✓B)	645.28 (✓)	2081.99 (✓)
0.6	60	10000 (✗)	10000 (✗)	2689.62 (✓B)	4078.71 (✓B)	10000 (✗)	10000 (✗)
0.8	80	10000 (✗)	10000 (✗)	10000 (✗)	10000 (✗)	2642.73 (✓B)	4079.05 (✓B)
1.0	100	10000 (✗)	10000 (✗)	10000 (✗)	10000 (✗)	10000 (✗)	10000 (✗)

Table 3.10 Trend observed by Query 1 and Query 2 in the fault tolerance experiment with and without fog functionality.

Probability Value	**% of total services getting broken**	**Query 1 and Query 2 With Fog Functionality**	**Query 1 and Query 2 Without Fog Functionality**
0	0	3 out of 3 times served	3 out of 3 times served
0.2	20	2 out of 3 times served (with backup service used 2 times)	2 out of 3 times served (with backup service used 1 time)
0.4	**40**	**3 out of 3 times served (with backup service used 1 time)**	**2 out of 3 times (with backup service used 1 time)**
0.6	**60**	**2 out of 3 times served (with backup service used 2 times)**	**1 out of 3 times served (with backup service used 1 time)**
0.8	80	1 out of 3 times served (with backup service used 1 time)	1 out of 3 times served (with backup service used 1 time)
1.0	100	Never served (User received an error message)	Never served (User received an error message)

de-merits of having a fog enabled system in such IoT scenarios. As visible from the plot in Fig. 3.18a, the number of times the user-requests were served during the breakdown of services almost trace-down each other, and only at 40% and 60% of services being broken, the fog assisted approach serves all the requests, while cloud centric approach gets totally disrupted there atleast once out of a total of 3 runs.

A possible explanation for this can be that in cloud centric approach there is only a single point of failure, while the fog assisted approach being distributed in nature, the points of



(a) Plot representing number of the times the user access query was served in three experimental runs of fault tolerance experimentation.

(b) Plot representing average service delivery time by the system for user access queries during fault tolerance experimentation of the system.

Fig. 3.18 Fault tolerance experimental analysis.

failure also get distributed, and thus is able to increase the overall fault tolerance of the system developed.

Also, a near usual-trend for service delivery time as observed during the times of non-disruption of services is observed during the breakdown of services as well i.e., the average service delivery time is less with the fog assisted approach and is more in cloud centric approach. The same has been shown in plot presented in Fig. 3.18b.

We also did an empirical mathematical analysis of such behaviour by the system using concepts of combination and probability theory. This has been presented in **appendix A**, and is a further possible extension for this work. With a more distributed setup coming from the MELD project, we plan to do a further analysis of the system behaviour continued from the empirical analysis as presented, and combine it with the concepts of Chaos Theory [180], [181]. This is part of the future direction of work coming from this chapter, and has also been included in Chapter 6.

Scalability: Scalability is an attribute that describes the ability of a process, network, software or organization to grow and manage increased demand [182]. It is the feature of a software solution to handle increased workloads. This can be larger data-sets, higher request rates, or a combination of both.

Usually at the production level, to handle such changes or to achieve scalability, there are two approaches that are used:

1. Vertical Scaling (Scale Up): Over here, scaling is done by adding more computational power (CPU, Memory) to an existing machine. Vertical scaling is often limited to the capacity of a single machine, and scaling beyond that often involves downtime in

order to add more computational power to the machine. Therefore, scalability with vertical scaling is limited.

2. Horizontal Scaling (Scale-Out): Over here, the scaling is done by adding more machines into the pool of resources.

In our use-case, the scalability trait of the developed software can arise in the following possible scenarios:

1. When there is a change in the amount of data. This is also referred to as load scaling in literature [183]. This refers to the case when the number of animals on the farm increase and all the other variables stay the same.
2. The other possibility is to have a change in the number of simultaneous requests coming to access the service, i.e., the scenario when there are increased requests to access the system at the same time. This is possible in large scale farms having various stakeholders. For e.g., there can be multiple workers on a farm who need to access the application (and maybe at the same time).
3. When there is a change in the number of computational nodes. This is also referred to as strong scaling in literature [183]. This refers to the case of having multiple fog and cloud nodes in the setup, with all other variables the same.

For scalability testing, we perform the experiment in fog assisted approach only, as for cloud-centric approach, it can be assumed that given that cloud has comparatively more resources than fog. So, in the scenario where everything is being deployed and executed from cloud, all the traditional cloud system based scalability traits will apply by default. The objective of the experiment here is to see how the proposed approach of leveraging fog computing is contributing to scalability.

We limited ourselves to scenarios 1 and 2 for the scalability testing experiment for the developed solution. Further, we limited ourselves to Query 2 as the user access request to access the developed system. We wrote custom scripts to generate the data as was generated by the real animals in the SmartHerd deployment for scenario 1 of the experimentation.

Fig. 3.19 gives a more insight visual representation of the scalability experiment. Table 3.11 and 3.12 further present more insight into the experimentation.

In the SmartHerd setup, we had 150 cows (or 150 sensors generating data), one fog node and one cloud node with the configurations as listed in Table 3.2. From the understanding of the scalability trait of a software system as listed above, we define scalability for the developed IoT solution as the ability to handle increased workload (either more data or requests) without adding additional resources to the setup and without spawning a new service instance for any of the microservices, and perform the scalability testing

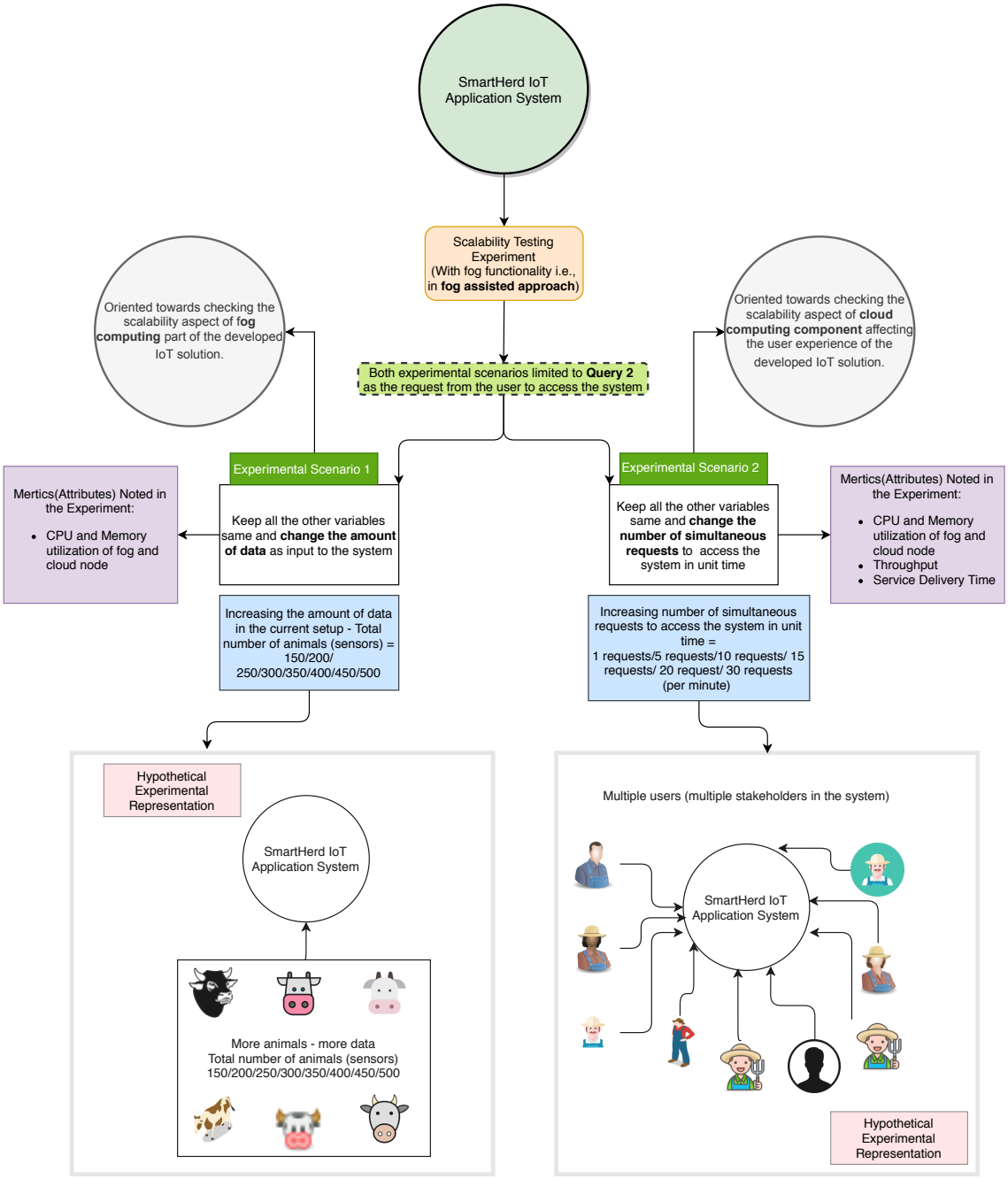


Fig. 3.19 Visual representation of the Scalability Experiment.

experiment accordingly. The objective of the scalability testing experiment here is to determine how the application performs with these possible changes. There are different possible attributes such as response time, throughput, CPU usage, Memory usage, Network usage that can be used as a metric while doing scalability testing.

We collected CPU usage, Memory usage, service delivery time and throughput as the metrics while doing scalability testing experiment on the developed IoT solution. It should

Table 3.11 Scalability experiment scenario 1 — changing the amount of data. This set of experiment was oriented towards checking the scalability aspect of fog computing part of the developed IoT solution.

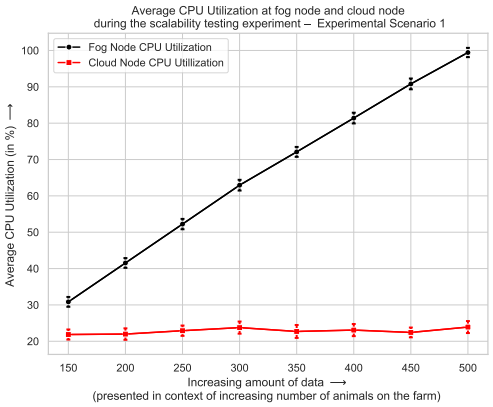
Amount of data (presented in context to the number of animals on the farm, which represent the corresponding number of sensors on the farm)	**Metrics Noted Down**
150	CPU and Memory Utilization of fog and cloud node
200	
250	
300	
350	
400	
450	
500	

Table 3.12 Scalability experiment scenario 2 — changing the amount of simultaneous requests. This set of experiment was oriented towards checking the scalability aspect of cloud computing component affecting the user experience of the developed IoT solution.

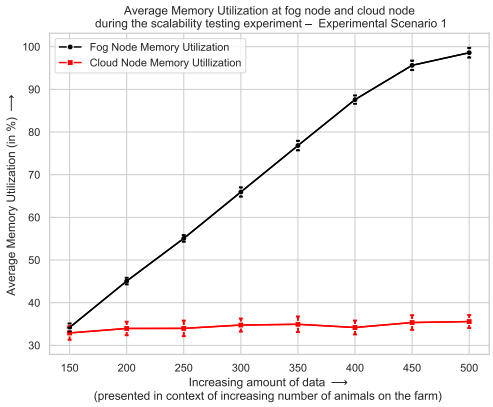
Number of simultaneous requests in unit time (*x* requests every minute)	**Metrics Noted Down**
1	CPU and Memory Utilization of fog and cloud node Service Delivery Time, and Throughput
5	
10	
15	
20	
25	
30	

be noted that throughput here is the measurement of number of requests processed in a unit time by the system. Its definition can differ from one application to another, for e.g. as in web application it is measured in number of user requests processed in a unit time whereas in database application it is measured in number of queries processed or transactions performed in a unit time.

The output of the scalability experiments have been presented in Fig. 3.20, 3.21 and 3.22. The whiskers in the plots represent the 95% confidence intervals for the values. It should also be noted that for each set (i.e., each row) in Table 3.11 and 3.12, the CPU and Memory utilization values are noted down for a period of 30 minutes and then the average value for the duration is used for making the plots. The analysis of the output obtained have been presented below:

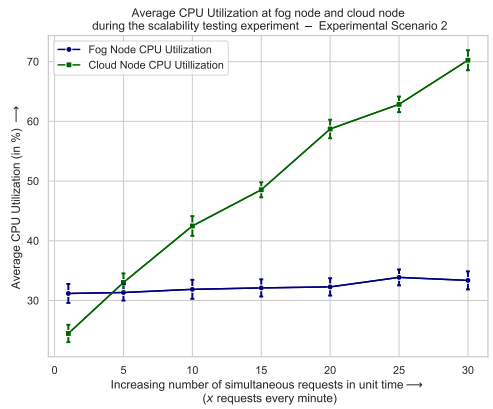


(a) Plot representing average CPU utilization at fog and cloud node during the scalability testing experiment in scenario 1.

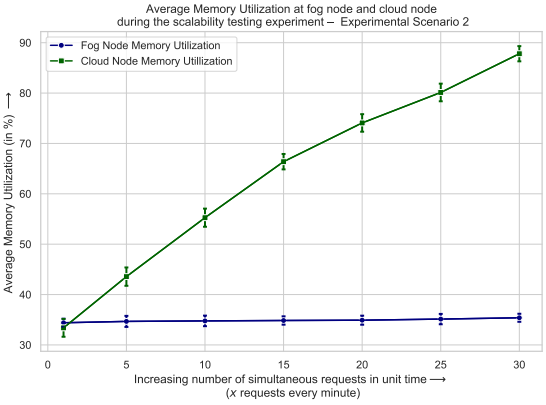


(b) Plot representing average memory utilization at fog and cloud node during the scalability testing experiment in scenario 1.

Fig. 3.20 Scalability experimental analysis — experimental scenario 1 — CPU and Memory utilization.



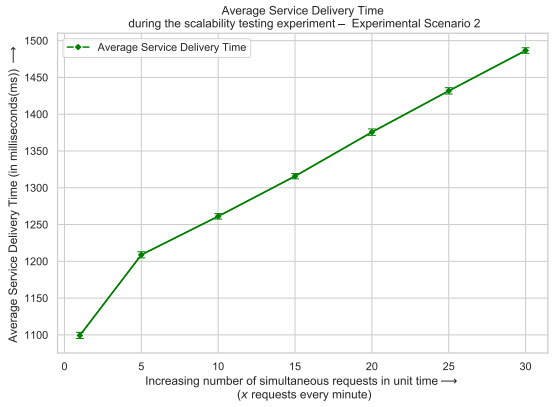
(a) Plot representing average CPU utilization at fog and cloud node during the scalability testing experiment in scenario 2.



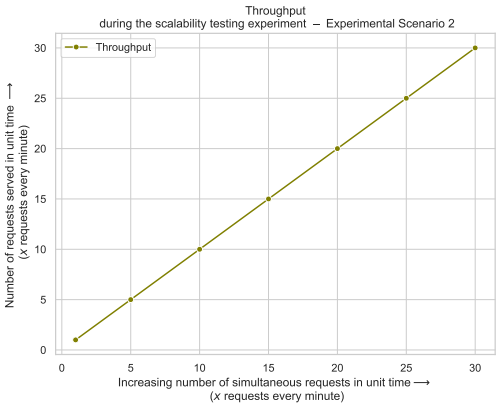
(b) Plot representing average memory utilization at fog and cloud node during the scalability testing experiment in scenario 2.

Fig. 3.21 Scalability experimental analysis — experimental scenario 2 — CPU and Memory utilization.

• **Experimental Scenario 1:** Fig. 3.20a and Fig. 3.20b presents the CPU and Memory utilization of fog and cloud node in the experiments. As visible from the plots, the CPU and memory utilization of fog node increases with an increasing amount of data, while there is very little effect on the cloud counterpart. As the data is processed at fog node and the outputs are sent to cloud node, thus there is an increase in the resource utilization. While at cloud node there is an increase in the incoming of the outputs obtained, but it is not as computationally intensive for the cloud component as it is for the fog, and thus the computing resource utilization stays almost constant here.



(a) Plot representing average service delivery time during the scalability testing experiment in scenario 2.



(b) Plot representing throughput during the scalability testing experiment in scenario 2.

Fig. 3.22 Scalability Experimental Analysis — Experimental Scenario 2 — Service Delivery Time and Throughput.

This can also be concluded from the results here that beyond 500 animals, which corresponds to the number of sensors sending data to fog node, there will be a need to have another fog node in the setup for the system to work well and deliver all of its functionalities. Hence, if the herd size exceeds 500, there will be a requirement to add more computing resources at the fog layer of this particular IoT deployment.

• **Experimental Scenario 2:** Fig. 3.21a and Fig. 3.21b presents the CPU and Memory utilization of fog and cloud node in the experiments. Over here, with an increase in the number of simultaneous requests coming to the service running on cloud, the CPU and memory utilization of cloud node increases, while there is a very little effect of it on the fog node. As all the user-requests are directed to cloud and served from there, and only the connected service if any is invoked at fog node, so the computing resource utilization at fog node stays almost constant here.

We further continued the experiment with increasing the number of requests every minute to check what are the maximum number of requests that are served by the service without invoking the auto-scale service i.e., without having another instance of the service. The value was found to be 46 requests per minute for the setup. At 46 requests per minute there will be a need to have multiple instances (at least two) of the user-access service.

Fig. 3.22a presents the average service delivery time for Query 2 in the experiment. With increasing number of simultaneous requests, the average service delivery time increases. This is potentially because of requests being queued before getting served, which results into an increase in the service delivery time.

Fig. 3.22b presents the number of requests that are served in unit time with increasing the simultaneous requests. For each request there is a separate connection that is established by the user-access service. The developed system is able to serve 46 simultaneous requests per minute as the threshold value, beyond which there is a need to have another instance of the service in place.

3.7.3 Machine Learning Objective of the Developed System

3.7.3.1 Cow Profiling

As the number of cows being analysed are significant in number, the first question is how to analyze them for an insightful analysis of their behaviour. Formally,

How to build robust cow profiles that are distinguishable by the learning model as lame and non-lame? Which parameter to use as baseline while building and comparing cow profiles?

For the system to differentiate between normal and anomalous behaviour due to lameness, we must first form profiles to characterize normal (non-lame) and lame behaviour in the herd. The most frequently used approach for this is to examine the activity level of lame and non-lame animals and study how these differ from the mean of the entire herd. But as it is known that outliers (i.e., a single element in sample being too high or low) can affect the mean value of sample; hence median or quantiles are sometimes taken as a better measure. To address this issue, we studied the relationship between the herd mean and the herd median. The results of this as presented in Fig. 3.23 show that these almost trace out each other for all the three activities (Lying time, Step count and Swaps). This is one of the features of a normal distribution, and therefore it would not matter whether the mean or median is used. Thus, we decided to use herd mean in our analysis.

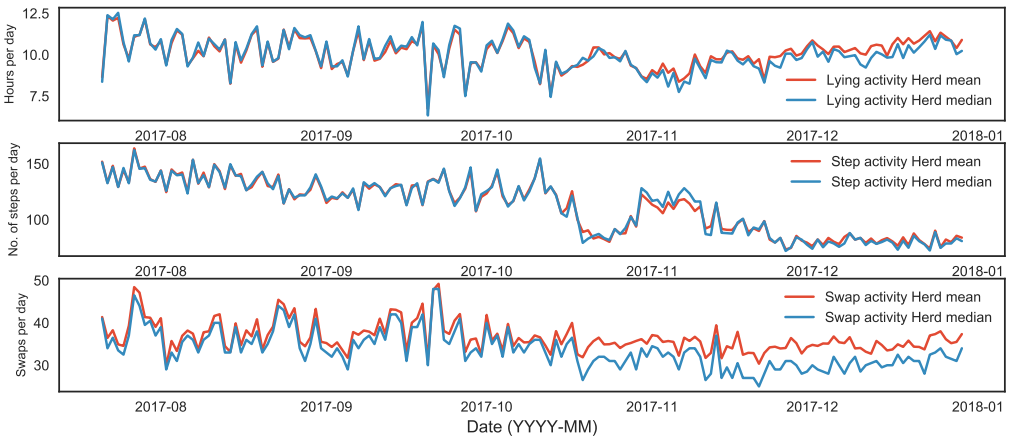


Fig. 3.23 Comparing the mean and median of the various animal activities.

A study [184] on animal behaviour analysis and association patterns of cattle shows that animals grazing within the same pasture can influence the movement, grazing locations, and activities of other animals randomly, with attraction, or with avoidance; therefore most of the animals will have their activity levels almost equivalent to the herd mean.

For such reasons, using herd mean as the baseline seems appropriate. Thus, any deviation from the herd mean should serve as a preliminary indicator for a sign of change in behaviour, which could potentially be lameness, among other reasons. Such an analysis eliminates the effects of external factors, as these will be largely affecting the herd as a whole. Further, the measure used to note the deviation in behaviour while forming Lame and Normal profiles of cows in the herd was Mean Absolute Deviation (MAD), and while comparing behaviour of individual cows with these formed profiles was Average Deviation.

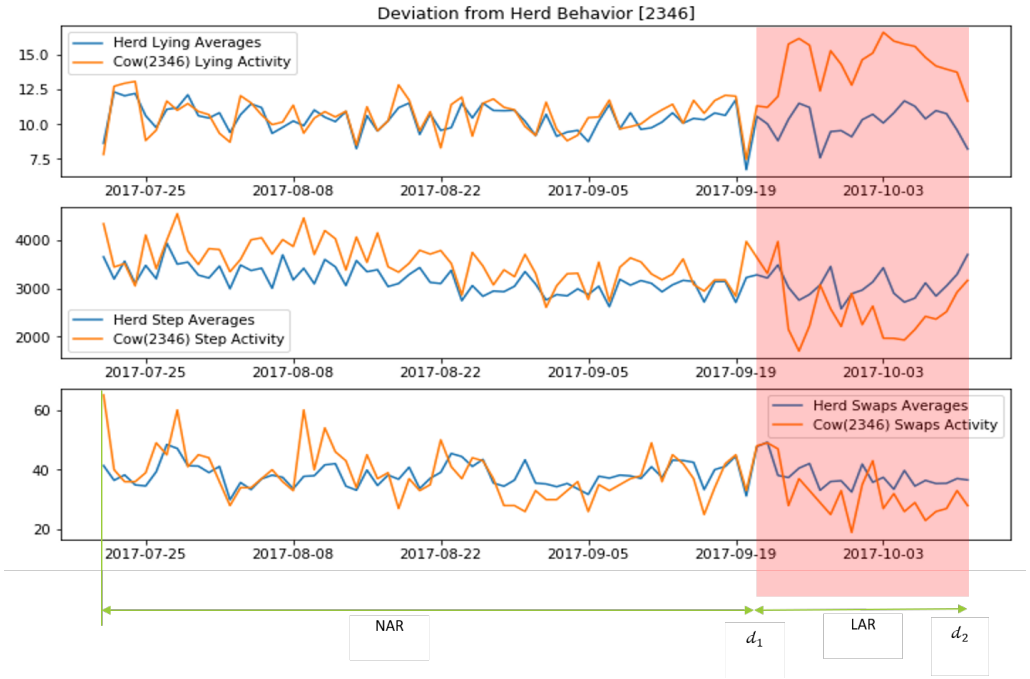


Fig. 3.24 Relationship between herd mean and cow activity for cow 2346, showing deviation in its behaviour from herd as it transitions into lameness.

We build a profile for each animal to characterize normal behaviour in a time window using activity based threshold clustering, details of which have been presented further in the next section (3.7.3.2). This helps us to define Lameness Activity Region (LAR, the period during which the animal is confirmed lame) and Normal Activity Region (NAR, the period during which animal is confirmed as non-lame), which later acts as ground truth input for the classification model for detecting lameness. An example of this has been presented in Fig. 3.24 for a random cow with ID 2346 in the herd.

Once a cow is identified as lame, we compare the herd mean for all the activities to that cow's activities and define a region $d_1 \leq D > d_2$, where d_1 is the day the activity starts

to deviate from the herd activity mean, d_2 is the day that cow is identified as lame. As lameness is a transition, we ascertain that the cow will remain lame after that until it is out of its lameness cycle. D is the entire duration between d_1 and the days after d_2 until the cow is out of its lameness cycle. It is the whole duration between d_1 to the last day when the cow was still lame. The values of d_1, d_2, and D will vary for each cow as some may have longer lameness cycles than others, and also depending on when the cow is identified as lame. This is motivated by the fact that lameness is a transition from normal behaviour to lameness and back, it will probably start before it is seen and even continue after treatment until the cow becomes normal again. Once we define the LAR, the rest of the graph is treated as the NAR.

However, by comparing the activity of each cow against the herd mean, we found out that not all animals behave the same way. Not all the animals in the herd had their activity tracing the herd mean — some had higher, some lower and some equal. This observation led us to our next decision in the analysis, which was to identify the clusters in the herd.

3.7.3.2 Clustering

Now given that the number of cows being analysed are significant in number, the intuitive step would be to somehow group them into smaller groups for the behavioural analysis. Formally,
Does each animal in the herd need to be treated separately i.e., treating each cow as a single experimental unit; or can some clustering technique be used to define clusters of animals that share similar features within the herd?

In this work, we tried two clustering methods to group animals into smaller groups that share the same features with in the herd. The first intuitive method was based on their **age**, and the second one which is **activity-based threshold clustering** came from a literature study and our observation on animal activities over a period of time. The age based clustering did not lead to the objective of early detection of lameness while the activity based clustering did help in making robust cow profiles, and helped with the objective of the developed solution

Fig. 3.25 presents the diagrammatic representation of the two clustering methodologies tried during the project, and further details on this has been provided below:

Age-based clustering : The total number of animals used to form clusters here were 147 as three of the animals were eliminated due to some health related issue. The age distribution of these 147 cows are as depicted in table 3.13. The first and most intuitive way of clustering the cows is in terms of age groups— young (age 2-5) and old (age 6-12). While this matches the intuition that the activity patterns of young cows are closer to the herd mean as opposed to older ones in the herd, there are crucial aspects missed out. For

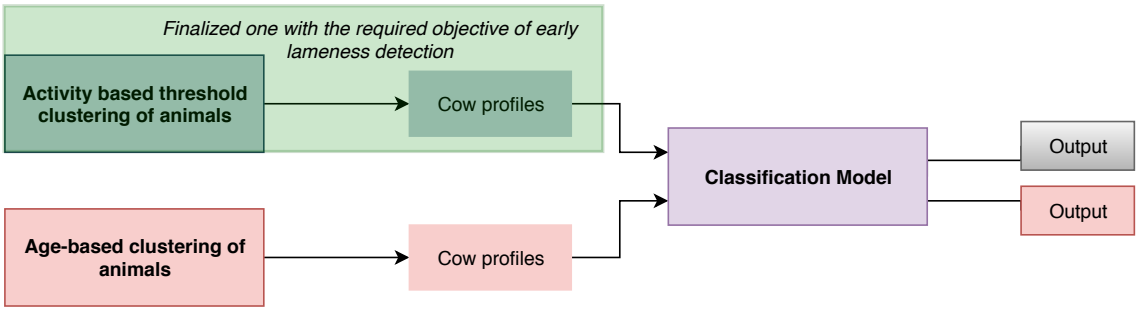


Fig. 3.25 Animal clustering methodologies tried in the project to make cow profiles, and the formed profiles passed to the classification model with the objective of early detection of lameness.

instance, the sample size in young category exceeds the old by 67%, and it is difficult to trace out the sub sets of animals that may behave differently in these binary groups. It thus excludes the outliers in each group and generalizes the classification as a whole, which leaves the true behaviour of a cow difficult to trace. The behavioural trend of young and old cows for the three activities is as shown in Fig 3.26 and Fig 3.27 respectively.

Table 3.13 Age Distribution of Cows in the Herd.

Age (in years)	Number of Cows	Cateogry
2	29	
3	31	Young cows (Age 2-5)
4	29	Number of young cows = 111
5	22	
6	9	
7	7	
8	7	Old cows (Age 6-12)
9	5	Number of old cows = 36
10	7	
12	1	
Total	147	147

Activity-based threshold clustering : The number of animals used to form clusters here and as presented as the final output from the project were 146 as three of the animals were eliminated due to health related issue as mentioned before, and one animal lost their tag during the experiment.

The same study [184] referred earlier in section 3.7.3.1 also shows that cattle in the same pasture are not treated as independent experimental units because of the potential confounding effects of the herd's social interactions. It also provides the insight that activity patterns of groups of cows with in the herd may have level of independence that is sufficient for analysing them as individual units under situations such as large herd size of around 53-240 cows. This means that smaller sizes herd (less than or equal to 40) don't exhibit any patterns of group formations within the herd while the larger herd

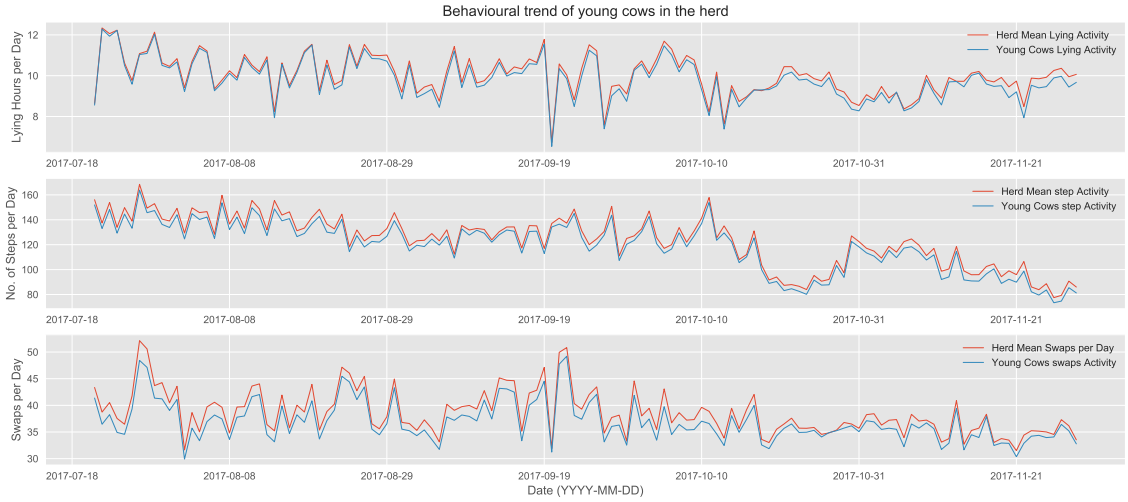


Fig. 3.26 Age based clustering of cows in the herd — Young Cows: Behavioural trend of young cows for the three activities in the herd.

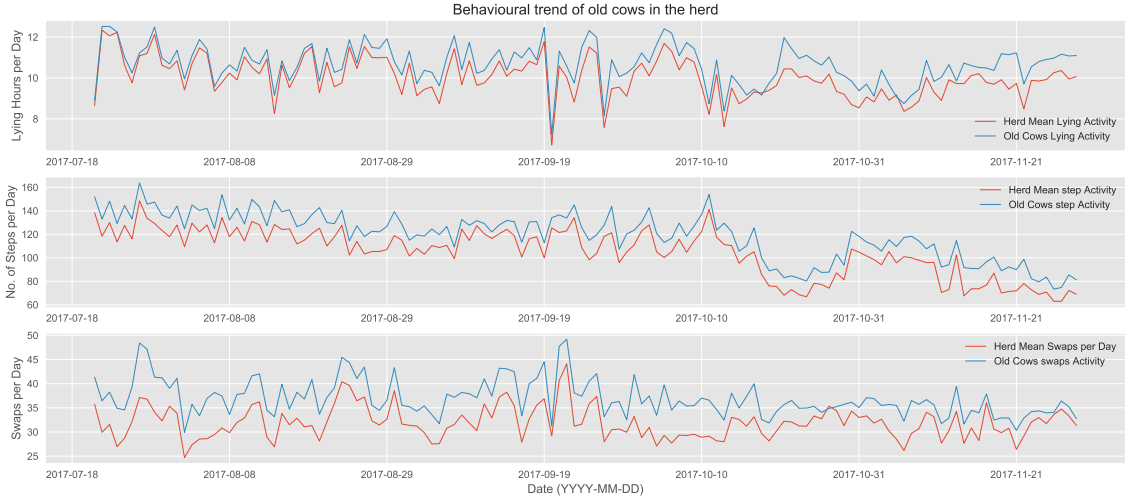


Fig. 3.27 Age based clustering of cows in the herd — Old Cows: Behavioural trend of old cows for the three activities in the herd.

sizes (53-240) show formations of groups within the herd. It should also be noted that the technology based automated smart solutions for animal welfare are more beneficial for farms with large herd sizes; one can assume that for small ones the farmer can manually keep track of each animal's welfare without much effort.

There can be no one-size-fits-all solution for determining the behaviour of an individual cow, considering the difference in the behaviour of sub-sets within the same herd. Not all cows would have the same levels of activity as others their age, or behave similarly. From our analysis and literature study, it was clear that a one-size-fits-all approach where it is assumed that all animals behave the same way, and all cows are treated as a single set (i.e., without any grouping) to detect anomaly in behaviour would not be efficient. There exist

sub-sets in the herd that share similar features, which once identified can be leveraged to fit the use-case as opposed to a one-size-fits-all solution. In our analysis, we found that even animals of the same age behaved differently and had different levels of activity.

Our clustering model is based on the observation that there are some animals in the herd whose activity levels (step count, lying time and swaps) were always greater than the mean activity value of the herd, while some whose activity levels were always less than the mean herd activity, and then there were others who traced the herd mean. It is also important to note that even when they became lame they had different activity levels depending on which category they belonged to.

To define a cluster, we define a window of size k days, and calculate MAD (Mean Absolute Deviation) between the cow activity and the herd mean for all the three activities.

$$C_{MAD} = \frac{\sum_{k=1}^{n} |H_m - C_i|}{k} \tag{3.1}$$

Here H_m is the herd mean within a defined window, C_i is the cow activity for activity i and k is the window size. We varied the values of k while testing the accuracy of the classification model and concluded that 14 days would be the optimal number of days to define a cluster. Based on MAD, we defined a threshold h. Now based on this threshold, and the following criterion, we define three clusters. If any two of the activity levels are below a certain threshold, then that animal is assigned into one of the below clusters:

- Active: These are animals in the herd whose activity levels are always higher than the herd mean. These have the mean deviation of any two of the activities greater than threshold h.
- Normal: These are animals in the herd whose activity levels always trace out the herd mean. These have the mean deviation of any two of the activities less than h but great or equal to zero.
- Dormant: These are animals in the whose activity levels are always lower than the herd mean. These have the mean deviation of any two of the activities less than zero.

The threshold was carefully chosen by a repetitive evaluation process, and was set to 1.7. The results presented in next section (3.7.3.3) have been derived with h value equals to 1.7. From our further analysis and investigation, we found that these clusters are dynamic in nature, i.e., the animals can migrate from one cluster to another in a time window. There can be a number of reasons behind this, we postulate because of age and weather at least, and perhaps other factors that affect the activity levels of the animals and the herd as a whole. Thus, it is the responsibility of the clustering model to re-cluster the animals

prior to feeding it into the classification model. The optimal time to re-cluster was found out to be about 2 weeks (14 days). This decision was made by continuously observing the movement of animals between different clusters, and finding the time frame of these movements. It should also be mentioned here that value of h was observed in both LAR and NAR period separately to ensure that it should not conflict with the lameness detection system being developed.

Fig. 3.28 shows the lying activity of the different clusters against the herd mean, and Fig. 3.29 and 3.30 depict the same for number of steps and swap count respectively.

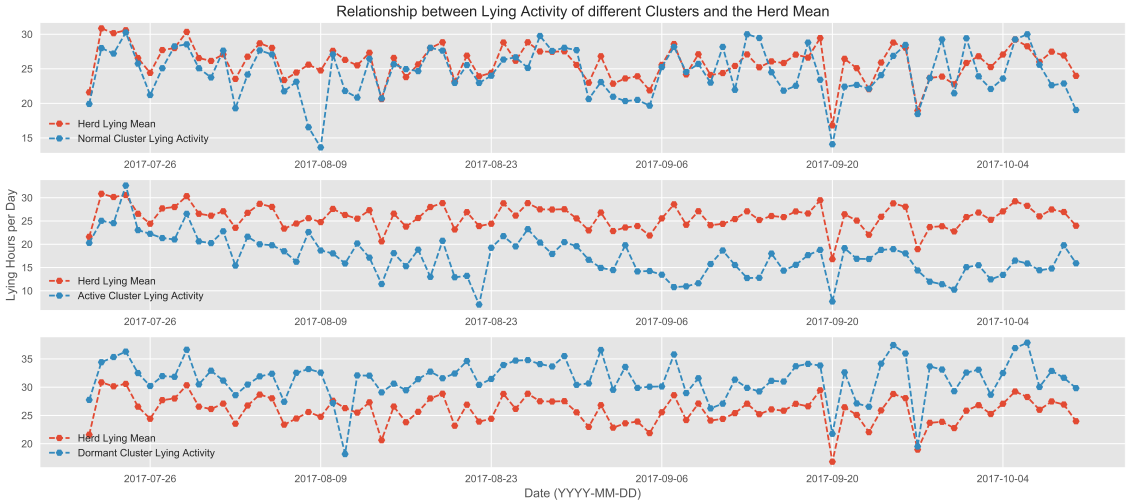


Fig. 3.28 Lying activity of the clusters against herd mean

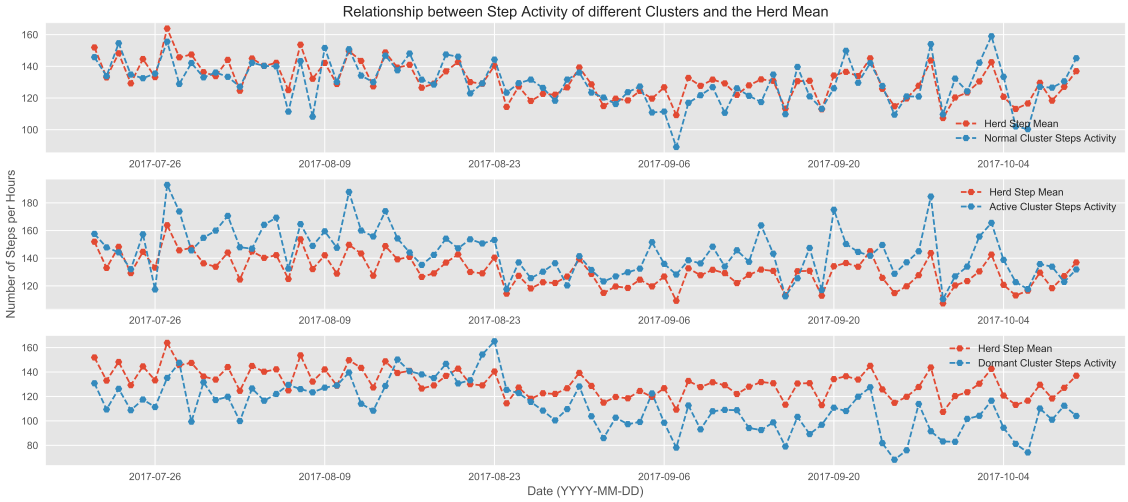


Fig. 3.29 Step activity of the clusters against herd mean

This method of clustering is unbiased of age, thus not only covers the outliers within each age group, but also depicts a more in depth analysis of the activities of each of the cows belonging to each age group, and the overall analysis of the herd activity.

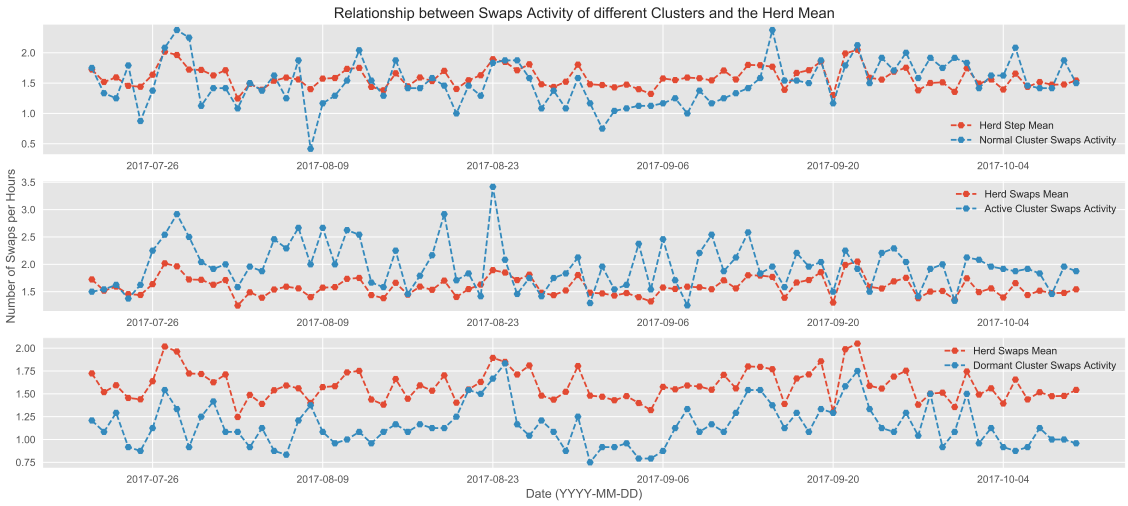


Fig. 3.30 Swap activity of the clusters against herd mean

As mentioned earlier that during the life span of the project, we started with 150 cows, then had 147 cows, and at a later final stage we had 146 cows in our analysis. Table 3.14 shows the distribution of these activity clusters as of final reporting of the project in June 2018.

Table 3.14 Distribution of cows in each activity clusters with 146 cows in the analysis.

Total number of cows in the analysis (as of June 2018)	**Active Cluster**	**Normal Cluster**	**Dormant Cluster**
146	25	109	12

3.7.3.3 Hybrid Machine Learning Model

Fig. 3.31 gives a quick overview of the SmartHerd pipeline from a machine learning objective of the developed solution. The aim of this representation is to provide an end-to-end representation of the system from data analytics perspective. It presents the end-to-end pipeline of the developed solution illustrating: 1) data collection from sensors, 2) observation of the herd by an animal expert for locomotion scoring, 3) translating the human observer's expertise into a machine learning based system leading to early detection of lameness in dairy cattle. Table 3.15 presents the locomotion scoring scale system used by the agricultural science student during animal observation in this study.

Fig. 3.32 presents the overall blended clustering and classification model for early lameness detection in dairy cows. As visible, the cows in the herd are clustered first based

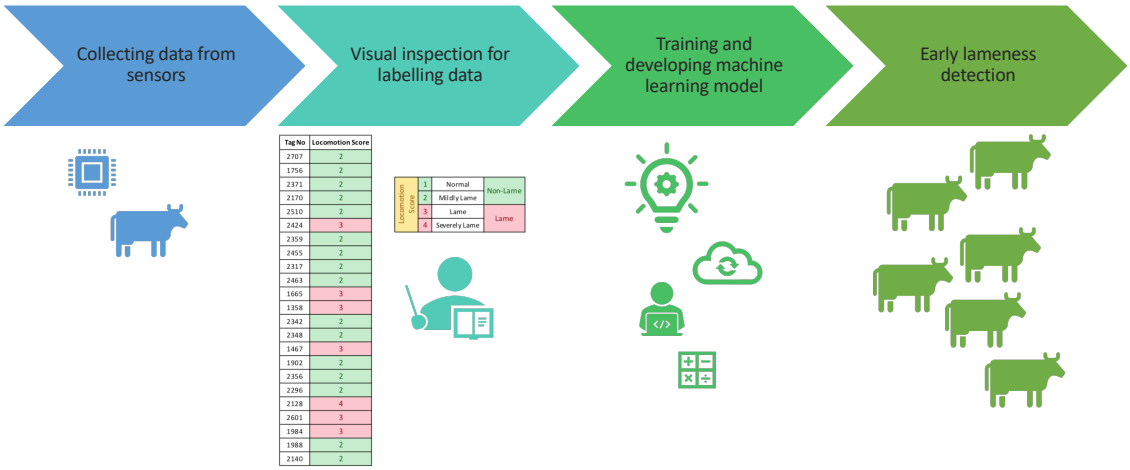


Fig. 3.31 A diagrammatic representation of the end-to-end pipeline of the developed solution illustrating: 1) data collection from sensors, 2) observation of animals by an animal expert to give locomotion score, 3) translating the human observer's expertise into a machine learning based system leading to early detection of lameness in cattle.

Table 3.15 Locomotion scoring scale system used by the agricultural science student while observing cows.

Locomotion Score	1 2	Normal Mildly Lame	Non-Lame
	3 4	Lame Severely Lame	Lame

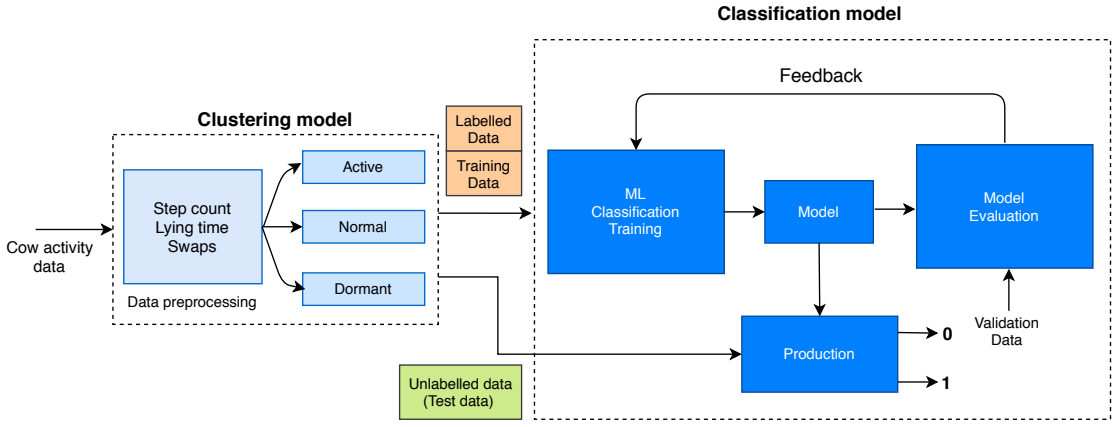


Fig. 3.32 Designed hybrid machine learning model for early lameness detection consisting of activity-based threshold clustering followed by the classification model.

on the threshold based activity clustering, and then for each of these clusters a model is

trained, and put in production after validation; so three separate models are made — each corresponding to one activity cluster.

Classification — early lameness detection : Classification algorithms belong to the set of machine learning algorithms that output a discrete value. Often, these output variables are referred to as labels, classes or categories. Classification problems with two classes are called binary classification problems, and those with more are referred to as multi-class. In our use-case scenario, the problem was written as a binary classification problem with Lame being the positive class and Non-lame as the negative class. The data split was as 80-20, i.e. 80% of data was used for model training and rest 20% was used for testing. The final classification algorithm used in this study was K- Nearest Neighbors (K-NN).

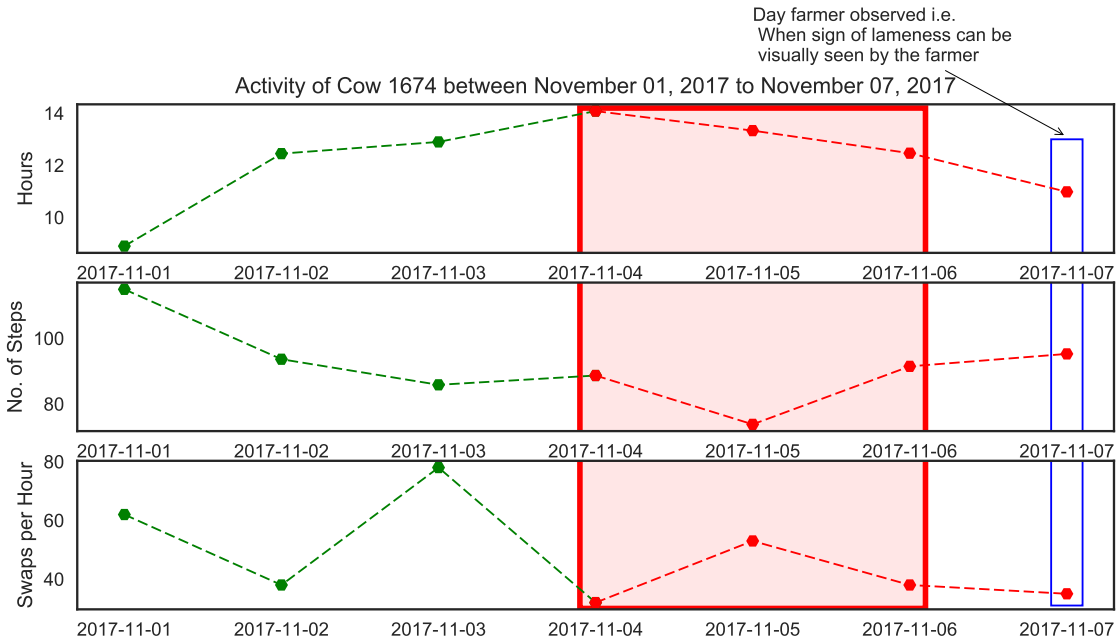
Accuracy of the developed system : We experimented on a number of sklearn [185] classification algorithms ranging from Support Vector Machine (SVM), Random Forest (RF), K- Nearest Neighbors (K-NN) and Decision Trees. We then went ahead with the K-NN base classification algorithm, as it was best balanced in terms of accuracy and early lameness prediction window as shown in table 3.16. It is also important to note that although a different model was trained and built for each of the three clusters (i.e., three classification models – one for each cluster), results reported (performance and accuracy) in this study are only for the normal cluster. This is because it was not possible to efficiently evaluate the other two clusters as testing data in these was very small (i.e., imbalanced for a proper evaluation).

Table 3.16 Lameness detection accuracy of the developed system.

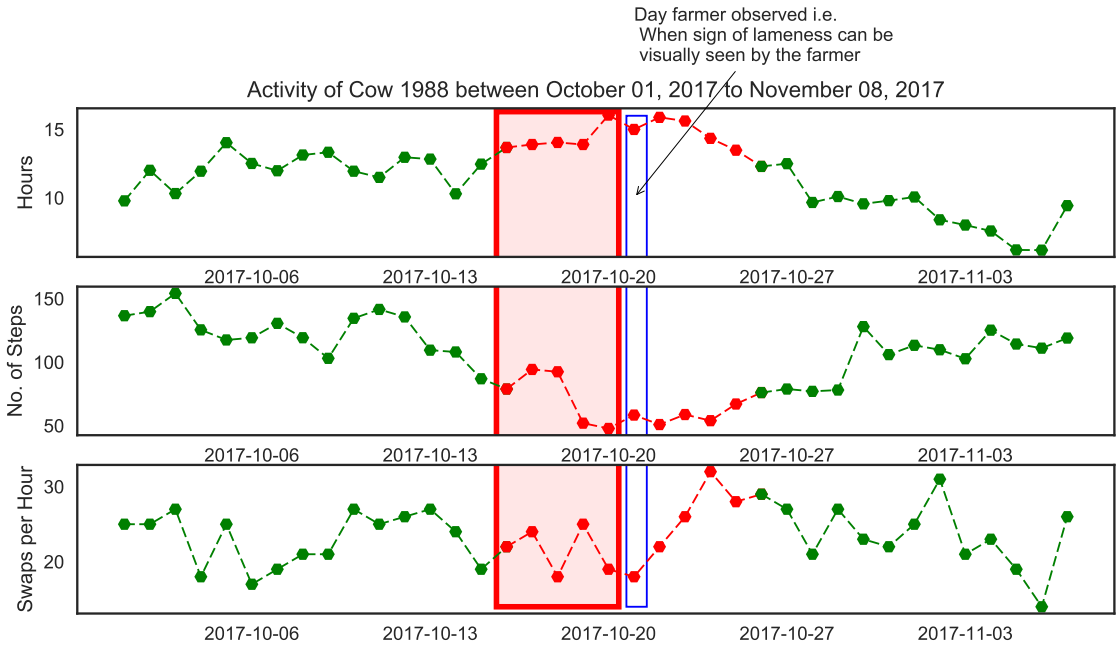
Classification Model	**Accuracy (in %)**	**Number of days before the visual signs of lameness appears**
Random Forest	91	1
K-Nearest Neighbors (K-NN)	87	3
Support Vector Machine (SVM)	61.5	2

K-NN: It has a number of parameters that should be fine-tuned in order to achieve the desired results. Among these, we evaluated different K-values (2-5), which is the number of neighbours to consider while assigning the nearest class. We set the distance metric to minkowski. The highest accuracy was obtained with $k = 2$ although this was over fitting the data. The accuracy of the developed system has been presented in table 3.16. Optimal results were obtained at $k = 4$ which gave an accuracy of 87% with 3 days before the visual signs could be seen as presented in table 3.17.

Early lameness detection assessment: Figure 3.33 shows two cases where the model was able to detect a cow being lame 3 days before its visual clues were available to the



(a) The highlighted red box shows the number of days the cow was lame but undetected and to only be visually seen on 07/11/2017 (highlighted and arrowed blue-box), and the beginning of red dots and dotted line shows that model detected it three days before on 04/11/2017.



(b) The highlighted red box shows the number of days the cow was lame but undetected and to only be visually seen on 20/10/2017 (highlighted and arrowed blue-box), and the beginning of red dots and dotted line shows that model detected it three days before on 17/10/2017.

Fig. 3.33 Early detection of lameness by the developed model and late observation by farmer.

Table 3.17 Different K-values and accuracy of the developed system.

K-value	Accuracy (in %)	Number of days before the visual signs of lameness are captured
2	91	1
3	89	2
4	87	3
5	81	1

farmer. The highlighted blue box shows the day when it was visually detected by the farmer or animal expert, and the start of the red points shows when the model detected the cow to be lame, and highlighted box shows the number of days for which the visual sign did not appear to be seen by the farmer or animal expert.

3.8 Summary and Conclusions

We have outlined the key design principles used in the development of our IoT solution aimed at early detection of lameness in dairy cattle. We present the critical decisions made and methodologies used in designing an end-to-end software system in fog-enabled IoT scenarios for our use-case. The key takeaways are as below:

- A distributed modular application architecture using microservices would be apt for design and development of IoT applications in fog computing environments.
- The results suggest that the proposed application design and development approach leveraging the fog computing paradigm is scalable, is able to efficiently utilize the computing resources available in the infrastructure, and also increases the fault tolerance and system-resilience of the overall system.
- A hybrid machine learning model such as one presented — activity based clustering combined with classification model — returns accurate results in detection of anomalies in animal behaviour for early detection of lameness as opposed to one-size-fits-all approach.
- Results clearly suggest that once monitored, the behavioural changes when animals are ill can be mapped to specific illnesses such as lameness in our use-case scenario.
- Many of these behavioural changes that occur before visual onset are extremely subtle, and difficult to detect in practice without technology.

- A careful coordination of computational resources along the technology path from sensor to cloud continuum is vital to the performance of such a system. Edge, fog and cloud resources each bring their unique input towards the functionality and performance of the overall IoT application system developed.

We believe that the insights from this study can contribute to the behavioural analysis of animals, and can help detect subtle changes in livestock behaviour before any clinical symptoms of disease are visible. This will lead to improved insights in animal behavioural analysis, and better practices for farmers. The wearable technology for livestock in conjunction with advanced machine learning methods has the potential for development of robust early warning systems to detect disease development early-on.

Chapter 4

A Framework for Efficient Placement of Components of an IoT Application in the Network Infrastructure Leveraging the Fog Computing Paradigm

4.1 Introduction

Since IoT aims to bring every object online, it constantly generates a huge amount of data that can overwhelm the storage systems and cause a significant surge in the application reaction time. With IoT into play, the near future will involve billions of interconnected IoT devices emitting large volumes of data streams for processing, leading to a momentous shift in the way applications are developed and deployed. Now, with this evolving scenario, there arises the need for a coherent approach of deploying these applications for an efficient utilization of the network infrastructure. While an application consists of various modules that run together with active inter-dependencies; traditionally, all these modules run on the cloud hosted in global data centres. With fog computing into picture, computation is dynamically distributed across the fog and cloud layer, and the modules of an application can thus be deployed closer to the source on devices in the fog layer.

With the increasing IoT deployments and the volume of impact increasing exponentially, a coherent approach of deploying these applications is critical for an efficient utilization of the network infrastructure. When deciding on where to deploy the application components over the continuum from things-to-cloud, application administrators need to find the best deployment, satisfying all the application requirements over the available computing resources. This chapter presents a Module Mapping algorithm for efficient utilization of

resources in the network infrastructure by efficiently deploying application modules onto fog and cloud resources for IoT applications.

This chapter is structured as follows: §4.2 presents the system architecture, mathematical model of the system, and formulates the problem, §4.3 presents the proposed approach and the methodology behind the development of the approach, §4.4 presents the experimentation, evaluation and validation of the proposed approach, and discussion on the experimental results, and §4.5 summarizes the work and contains the concluding remarks.

The work presented in this chapter has been disseminated in the following publications: **P3 - IM 2017** [12], **P2 - IEEE/ACM SEC 2016** [11], **P1 - CF Procedia 2016** [10].

4.2 System Architecture and Problem Formulation

This chapter addresses second research question (RQ2). It addresses:

How to efficiently deploy components of a multi-component IoT application onto computing resources available in the infrastructure leveraging the fog computing paradigm?

The remainder of this section presents the system architecture, simulation platform used, application modeled, and mathematical model of the problem being addressed here by the proposed framework.

4.2.1 System Architecture

A generic three tier IoT-fog-cloud architecture illustrating the distributed data processing has been shown in Fig. 4.1. While the end devices come under the first tier, the fog and the cloud layer comprise the second and the third tier respectively, together forming the three tier architecture. Each tier can be mapped to support a specific component of the application, which is further elaborated and worked upon in this section.

In the architecture, any element in the network that is capable of hosting application module(s) is considered a fog device. As discussed earlier in 2.4.1 and depending on the use-case, it may be some other common device, such as small server, or routers, access-points, or gateway etc. Amongst the available networking devices in the network infrastructure, the devices of prime consideration to us here are the gateways that connect the devices in the bottom most layer (Tier 1/IoT layer) to the Internet. While the fog layer can be conceptualized as that comprising of all the aforementioned devices, in practicality, the gateways have emerged as the key constituents of the fog, for the reason that they support/enable protocol conversion across different network segments, and are thus least conservative about additional requirements and constraints. Thus, the gateways that are

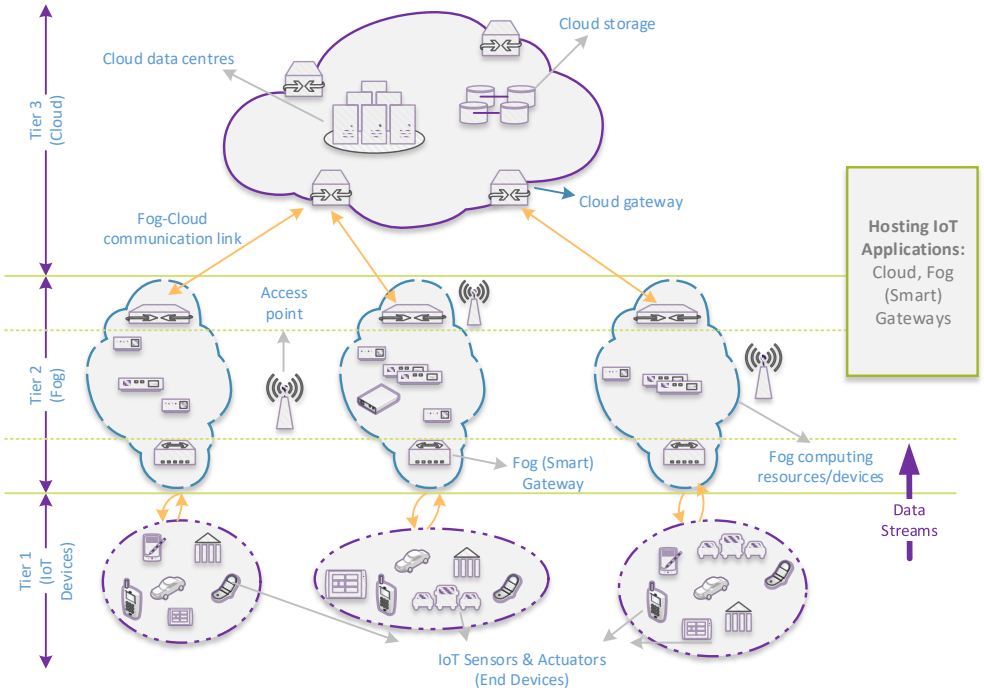


Fig. 4.1 Three tier IoT-fog-Cloud architecture illustrating distributed data processing [12].

also additionally working as fog devices in the network are often termed as Smart Gateways and are present in Tier 2 of the architecture as illustrated in Fig. 4.1

Each network node has a specific computing capability, which in turn forms (and contributes to) the resources available in that layer. While the devices in the fog layer have a defined and limited computing capability owing to a hardware constraint of the resources each device has, the cloud, on the other hand has corresponding resources realized in terms of multiple virtual machines (VMs), each configurable to a specific configuration which tells the computing capability of that VM. This is owing to the fact that the overall computational/resource capacity of the cloud, as a whole, is tremendously more than that of a typical device available in fog layer, so much that in comparison to the fog devices, the cloud is visualized as being limitless. Thus, the complete set of resources comprise of the devices at the fog, and VMs at the cloud.

4.2.2 Simulation Platform Used and Application Modelled

Simulation Platform: The simulation platform used in prototyping the IoT application, fog computing environment, and algorithm development is iFogSim [62], which is a toolkit developed by Gupta et. al [62] over the CloudSim [141] framework for modeling and simulating IoT, edge and fog computing environments. It uses the Distributed Data

Flow(DDF) model and Directed Acyclic Graph (DAG) representation while simulating any application scenario in fog computing environment.

Application Modelled: The application modelled is motivated from realistic scenarios like health care [131] and latency-critical gaming [130]. The application was formulated, modeled and deployed in iFogSim. The application works on the Sense-Process-Actuate model, where the information collected by sensors is emitted as data streams, which is processed and acted upon by application modules running on fog and cloud layer, and the resultant commands (or outputs) are sent to the actuators. The other model, which is Stream Processing model, is where a network of application modules running on fog and cloud layer continuously process data streams emitted from sensors; and information mined (or aggregated) from the incoming streams is stored in cloud hosted in a data centre for large-scale, long-term and complex analytics. The Stream Processing model is considered as a subcategory of the Sense-Process-Actuate model. These models can, however, be extended to cater use-cases other than IoT applications as well.

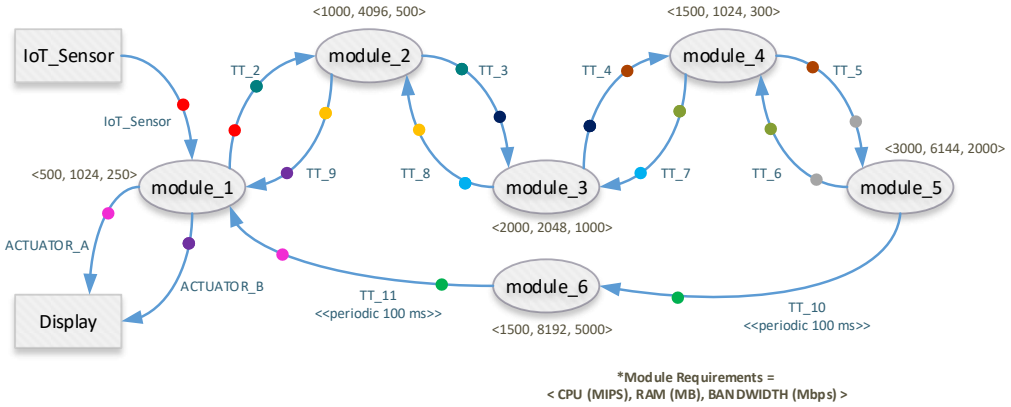


Fig. 4.2 Directed Acyclic Graph (DAG) of the application deployed, with relevant tuples indicating the values used for simulation of the algorithm. The number of modules as well as sensors/actuators are synchronous to the needs of a typical IoT application, and may vary as per the use case and application size; the modules can be conveniently linked to various logics to convey the various stages of application processing, as are the ones conveyed by the color pair encoding here [12].

As shown in Fig. 4.2, the DAG of the application modeled in the simulation consists of six application modules— module_1 to module_6. IoT_Sensor represents an IoT sensor, which emits tuples of type IoT_Sensor to module_1. Display is an actuator, which is designed to respond to changes in the environment captured by the sensor. Tuples form the fundamental unit of communication between entities in the IoT Ecosystem, and are indicated over the edges in the DAG. The colored dots signify tuple mapping, as in, for

example, an incoming tuple of type TT_3 on module_3 will result in an output tuple of type TT_4.

Note that module_1 of the application needs to be placed (and run) on the end devices (Device-X-X in topology in Fig. 4.3, and IoT_Sensor in Fig. 4.2) to ensure that the user base, or the source application modules, are co-located with fog devices in the fog layer. The actionable (sensor and actuator) are connected to these devices running module_1 on them.

4.2.3 Mathematical Model

Infrastructure: The computational capacity (or resource capacity) of a network node can be represented as a set of three narrowed-down attributes, namely CPU, RAM and Bandwidth. However, it is to be noted that the proposed algorithm is scalable even on adding more number of attributes for a node, and more of them (like storage capacity) can be included if need be. Thus, if n_i represents a network node i in the infrastructure, the capacity of the said node is represented as:

$$Cap(n_i) = < CPU_i, RAM_i, Bandwidth_i > \tag{4.1}$$

The set of all computing resources available with in the infrastructure is given by N.

$$N = \{n_i\} \tag{4.2}$$

N can be divided into two mutually exclusive subsets— N_F and N_C as below:

$$N_F = Set\ of\ network\ nodes\ in\ fog\ layer \tag{4.3}$$

$$N_C = Set\ of\ network\ nodes\ in\ cloud\ layer \tag{4.4}$$

$$N_F \cup N_C = N \tag{4.5}$$

$$N_F \cap N_C = \phi \tag{4.6}$$

Application Design and Architecture: The applications developed for deployment in this chapter are based on the DDF model [87]. Distributed computing environment calls for distributed components, which would give better results with multi-component applications, for which DDF is one of the best approaches available.

As shown in Fig. 4.2, the application is modeled as a DAG, with various application modules constituting the data processing elements. In context of analytics applications such as stream analytics or event based analytics, these modules are usually termed as application operators. In the DAG so formed, the vertices represent the various modules of the application, and the edges represent the data dependencies between them. These

modules perform processing on the incoming data, and the edges connect the output of one module to the input of another, representing the flow of data between the modules.

In mathematical notation, the DAG $\mathscr{G}$ of an application consists of vertices (V) and edges (E) and is written as follows:

$$\mathscr{G} = \langle V, E \rangle \tag{4.7}$$

Each module of the application has a requirement represented as a set of three attributes, namely CPU, RAM and Bandwidth. The proposed algorithm, though, is similarly scalable even on adding more number of attributes as requirements, and thus more of them can be included if need be. Thus, if v_i represents an application module i in the application, the requirement of the said module is represented as:

$$Req(v_i) \ = < CPU_i, RAM_i, Bandwidth_i > \tag{4.8}$$

The set of all modules of the application is devoted by V.

$$V = \{v_i\} \tag{4.9}$$

An edge originating from an application module v_i to another application module is denoted by e_i, and indicates data flow in an application. The set of all edges in the DAG of an application is denoted by E.

$$E = \{e_i \mid e_i = \langle v_i, v_j \rangle\} \quad \forall v_i, v_j \in V \tag{4.10}$$

There are two types of edges possible in a DAG— periodic and event based. Tuples on a periodic edge are emitted regularly at the specified interval; whereas in event based, a tuple is sent out if the source module of the edge receives an incoming tuple, and the defined selectivity model (fractional selectivity, in our case) allows its emission.

A Module Mapping function $\mathscr{M}$,

$$\mathscr{M} : V \rightarrow N \tag{4.11}$$

indicates the network node on which the application module is placed during the application deployment, such that it meets the following:

$$\forall (v_i, n_i) \in \mathscr{M} \quad \Big| \quad \begin{array}{c} \Longrightarrow Req(v_i) \leq Cap(n_i) \\ \forall v_i \in V \\ \forall n_i \in N \end{array} \tag{4.12}$$

While traditionally all the application modules were placed on the cloud, this brought in a tremendous network cost, in addition to a high application response. With the proposed

Fog-Cloud Placement approach, these application modules can be distributed across the fog and the cloud resources based on meeting the module requirements and network capacity constraints as shown in the above equation (4.12).

We further proposed to identify the DAG of the application as having static and dynamic characteristics as follows:

- *Static Characteristics:* These are those which we expect developers to provide, and remain invariant over time for an application— such as data (tuple) emission rate of sensors, data processing rate (selectivity model) of application modules, etc.
- *Dynamic Characteristics:* These are those which come into play once the application has been deployed on the network infrastructure. These are dynamic run time characteristics of fog and cloud resources (which are network nodes in the IoT ecosystem)— such as their network connectivity.

In the present study as shown in the chapter, the static characteristics of the application DAG are presented/studied, the results from which can further be translated to address the dynamic characteristics. Further, addressing dynamic characteristics of DAG has been included in the scope of future work discussed in the last chapter 6 of the dissertation.

4.3 Proposed Fog-Cloud Placement Approach and Algorithm Design

To enable resource aware placement of application modules onto the computing resources, we propose three integrated algorithms that have been presented in detail in this section.

4.3.1 Working of Algorithm

Algorithm 4.1 is the ModuleMapping algorithm, which enables fog-cloud placement. It returns the efficient mapping of modules of an application onto a network infrastructure. Taking the set of network nodes N and set of application modules V as input, it first sorts the network nodes and modules in ascending order as per their capacity and requirement respectively. A key-value pair corresponding to network node as key and application module as value is then created.

The control loop of the algorithm (for loop/line5) runs for all the modules of the application that need to be placed, and calls the function LOWERBOUND (Algorithm 4.2) in each iteration, which searches for the eligible network node meeting the requirement of the module (constraint specified in equation 4.12). The requirement check is ensured by COMPARE function (Algorithm 4.3), and when an eligible network node is found, the

Algorithm 4.1 ModuleMapping Algorithm: Fog-Cloud Placement

Input : Set of Network nodes N and Application modules V
Output : Mapping of modules on to network nodes

```
1:  function MODULEMAP(NetworkNode nodes[], AppModule modules[])
2:      Sort(nodes[]),Sort(modules[]);                          ▷ in ascending order
3:      Map < NetworkNode, AppModule[ ] > moduleMap;   ▷ Creates Key-Value Pair
    with Network Node as Key and AppModule as Value
4:      int low = 0, high = nodes.size-1, start;
5:      for  start =0 to modules.size do
6:          int i=LOWERBOUND(nodes[],modules[start],low,high);
7:          if (i != -1) then
8:              moduleMap.insert(nodes[i],modules[start]);
9:              Cap(node[i]) = Cap(node[i]) - Req(modules[start]);
10:             Sort(nodes[]);                                  ▷ in ascending order
11:             low = i + 1;
12:         else
13:    moduleMap.insert(nodes[nodes.size-1],modules[start]);
14:         end if
15:     end for
16:     return (moduleMap);
17: end function
```

corresponding key-value pair entry is added into the result (moduleMap). This way it iterates from fog nodes to cloud nodes, first placing the modules on eligible nodes in fog layer, and once the nodes in fog layer are exhausted or if there is no eligible node in the fog layer, only then it places the corresponding module on cloud.

4.3.2 Time Complexity Analysis

The sorting of the set of network nodes and application modules takes $O(|N| * log\,|N|)$ and $O(|V| * log\,|V|)$ time respectively. The LOWERBOUND function is called for all the modules of the application, and uses the principle of binary search as its basis for searching for the eligible network node for module placement, thus giving us the time complexity of $O(|V| * log\,|N|)$. An additional sorting is required after updating the network node selected for the placement of module (line 10, Algorithm 4.1) which gives us another time complexity of $O(|N| * log\,|N|)$. Thus the overall time complexity of the solution is $O((|N| + |V| + |N| * |V|) * log\,|N| + |V| * log\,|V|)$.

If $|N|$ is much greater than $|V|$, then the upper bound becomes $|N|*|V|$ and the complexity becomes $O(|N| * |V| * log\,|N|)$.

Usually, the brute force solution (i.e., searching through all possible combinations (2^N) and returning the best one) to such problems tends to be NP-hard [186], [187], and thus

Algorithm 4.2 LowerBound Algorithm - Algorithm used for Search

```
1:  function LOWERBOUND(NetworkNode nodes[], AppModule module, int low, int high)
2:      int length = nodes.size, mid = (low + high) / 2;
3:      while (True) do
4:          NetworkNode x = node[mid];
5:          if COMPARE(x, module) == 1 then
6:              high = mid-1;
7:              if (high<low) then return mid;
8:              end if
9:          else
10:             low = mid + 1;
11:             if (low>high) then
12:                 return((mid<length-1)?mid+1:-1);
13:             end if
14:         end if
15:         mid = (low + high) / 2;
16:     end while
17: end function
```

Algorithm 4.3 Compare Network Node and Application Module

```
1: function COMPARE(NetworkNode a, AppModule b)
2:     if (a.CPU≥ b.CPU && a.RAM≥b.RAM && a.Bandwidth≥b.Bandwidth) then
       return 1;
3:     end if
4:     return -1;
5: end function
```

we present the heuristic approach to the problem, which contributes to a logarithmic time complexity.

The problem of finding solution mappings between application components in G and available computing resources in N, and minimizing/maximizing objective function if any is NP-hard [186]. The perfect solution to such problems require evaluating all possible solutions in worst case scenario i.e., assume the application is made up of m components and the total number of computational resources in the infrastructure are n, then n^m different candidate solutions are possible. This kind of complexity can be managed by using heuristics that allow to find sub-optimal solutions with low complexity, such as the one presented by the proposed approach.

4.4 Experimental Evaluation and Validation

To test the proposed algorithm, the application was run on network topologies supplied by us as JSON (JavaScript Object Notation) file(s). The scenario has been varied over three network topologies with different workloads respectively, the graphical view of one of which as generated by iFogSim is shown in Fig. 4.3. The experiment was iterated on topologies with 2, 4 and 6 fog gateways, each having two devices per fog gateway. The experimental network configurations used in iFogSim can be found in table 4.1, 4.2 and 4.3.

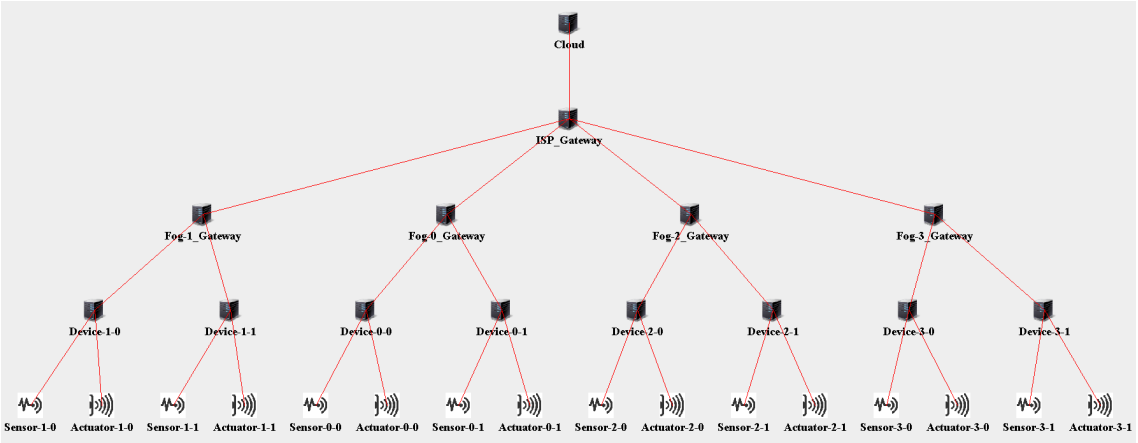


Fig. 4.3 One of the network topologies used for deployment iteration. The simulation has been varied over three such topologies with varied workloads, but essentially the same standardized network structure [12].

4.4.1 Experimental Setup, Configurations and Simulation

Table 4.1 Experimental network configurations used in iFogSim.

Between		**Latency (ms)**
Cloud	ISP_Gateway	200
ISP_Gateway	Fog-X-Gateway	25
Fog-X-Gateway	Device-X-X	5
Device-X-X	Sensor	2
Device-X-X	Actuator	3

The proposed fog-cloud placement approach (ModuleMapping algorithm) was compared with the traditional cloud-based placement approach in terms of application latency (response time), network usage and energy consumption; various metrics reported by iFogSim for the modeled application using both the placement approaches were collected.

Table 4.2 Experimental network configurations used in iFogSim.

Devices in Network Infrastructure	Upstream Capacity (Mbps)	Downstream Capacity (Mbps)	RAM (MB)	CPU (MIPS)
Cloud	1000	10000	40960	40000
ISP_Gateway	10000	10000	8192	10000
Fog-X-Gateway	10000	10000	6144	8000
Device-X-X	100	250	2048	4000

Table 4.3 Experimental network configurations used in iFogSim.

Tuple Type	Tuple CPU Length (MIPS)	Network Length
IoT_ Sensor	3000	500
TT_2, TT_3, TT_4, TT_5	6000	500
TT_6, TT_7, TT_8, TT_9	1000	500
TT_10, TT_11	1500	1000
ACTUATOR (A/B)	2000	500
#The average tuple emission rate of a sensor is 10 milliseconds, specified by a deterministic distribution.		

The results of the simulation (Fig. 4.4, 4.5, 4.6) demonstrate an immensely favorable impact on network usage, application latency (response time) and energy consumption in the proposed placement approach on all 3 network topologies used.

4.4.2 Results and Analysis

In this section we present the analysis of the results obtained by using the proposed approach as shown in Fig. 4.4, 4.5, 4.6.

• **Network Usage:** As shown in Fig. 4.4, there was noticed a staggering decrease in the network usage with the proposed fog-cloud placement approach. The reason behind this is that as now the application modules are being placed closer to the source of data, and hence the communication being done in the network decreases as visible from the plot in Fig. 4.4. While in the traditional cloud-centric approach, all the application modules are placed on the cloud, and all the tuples have to be transmitted across the things-to-cloud spectrum to be processed by the application, and hence more network-usage compared to the proposed fog-cloud placement approach.

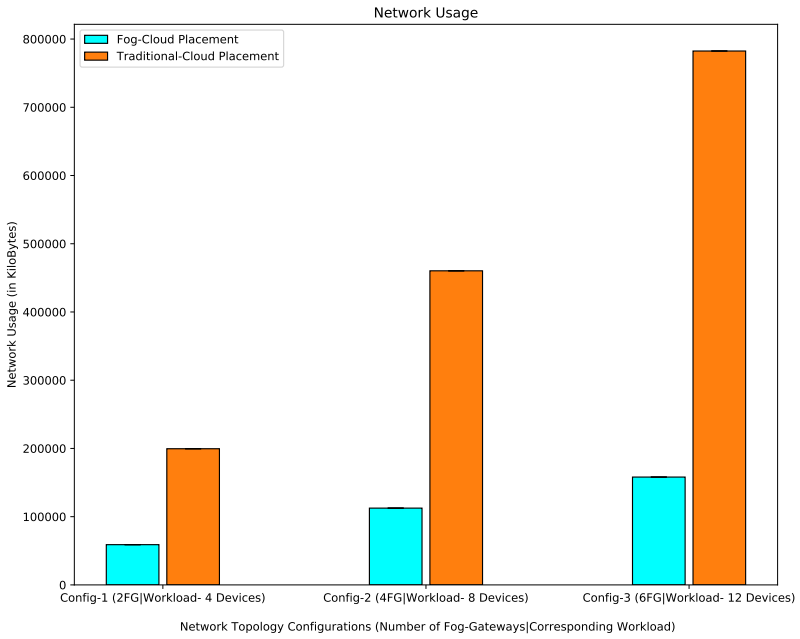


Fig. 4.4 Staggering decrease in network usage via proposed approach.

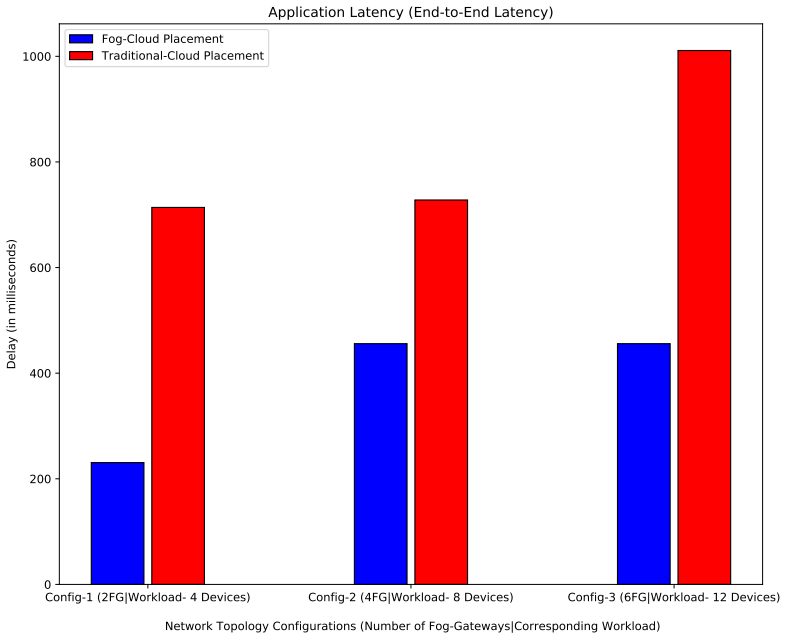


Fig. 4.5 Huge effect of efficient module mapping on end-to-end latency.

• **Application Latency:** There was also a huge effect of efficient module mapping on end-to-end latency, with highly favourable results towards fog-cloud placement as per the designated approach as shown in Fig. 4.5.

With the proposed fog-cloud placement approach the application-modules are being placed on fog and cloud resources both, instead of just being placed on cloud entirely. This leads to an overall decrease in the response time of the application as visible from plot in Fig. 4.5.

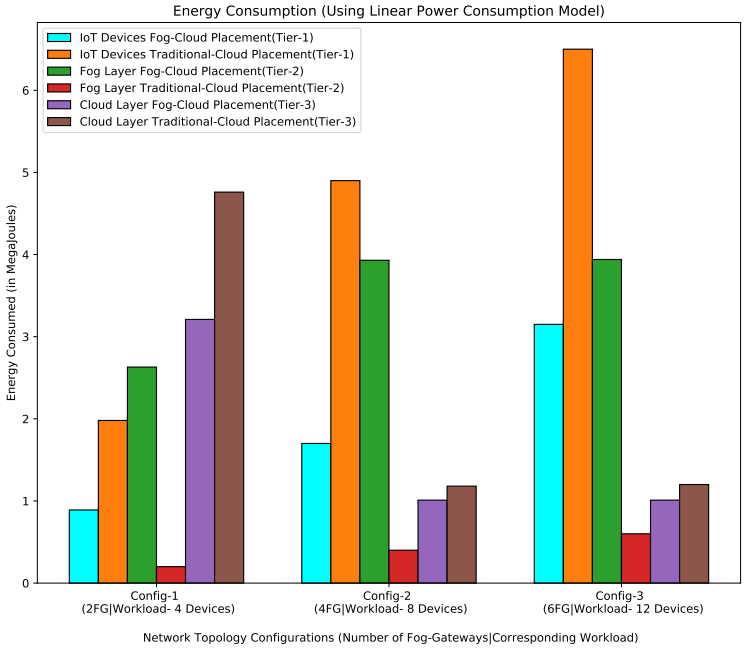


Fig. 4.6 Variation of energy consumption in both the placement approaches— Creating a balance between energy consumption in the cloud (at cloud data centres) and in fog layer (fog devices) by spreading computing across the network. The energy consumption in the cloud data center decreases when application modules are placed on fog devices using the proposed Fog-Cloud Placement approach compared to the Traditional-Cloud Placement strategy.

• **Energy Consumption:** Fig. 4.6 shows the variation of energy consumption in both the placement approaches, where we look towards creating a balance between energy consumption in the cloud (at cloud data centres) and in fog layer (fog devices) by spreading computing across the network via the proposed approach. By using the proposed approach, the application modules are placed on the computing resources available along the things-to-cloud continuum rather than being placed all together in the cloud. The energy consumption in the cloud data center decreases when application modules are

placed on fog devices using the proposed Fog-Cloud Placement approach compared to the Traditional-Cloud Placement strategy as visible in Fig. 4.6 .

The key observations here are as follows:

- Energy consumed by devices in IoT layer (Tier-1) is less in fog-cloud placement approach as compared to traditional cloud placement approach. A possible explanation for this can be that data tuples from devices are being processed just one hop away. Since application modules are being placed on fog nodes via the proposed approach, this leads to decrease in energy consumption of devices required for communication. While on the other hand in the traditional cloud-centric approach, the data tuples still have to be transmitted from the device to the other end of the infrastructure in cloud, and results in more energy consumption in the transmission process.
- Energy consumed in fog layer (Tier-2) is more in fog-cloud placement approach as compared to traditional cloud placement approach. The reason behind this is that in the proposed approach the application modules placed on fog nodes lead to more energy consumption with data tuples being processed there, while in the cloud-centric approach the fog nodes just act as the usual gateways to send the received tuples to cloud for processing without performing any computation on them, and thus less consumption of energy.
- Energy consumed in cloud layer (Tier-3) is more in traditional cloud placement approach compared to fog-cloud placement approach. In traditional cloud-centric approach all the application modules are hosted completely in the cloud, and thus all the computation happens entirely there. While with the proposed fog-cloud placement approach, some modules are placed on fog nodes and thus comparatively less energy consumption by cloud node in the proposed approach compared to the traditional cloud-centric approach.

4.5 Summary and Conclusion

The aim of second research question (RQ2) was to build an application deployment strategy for fog computing environments. We presented the benefits of having a fog enabled application deployment strategy, which results into efficient utilization of computing resources in the network infrastructure by means of efficiently deploying application modules onto fog and cloud computing resources for IoT applications. This shows the impact of the evolving computing paradigm towards solving the problem of latency in time critical IoT applications, while also accounting for the pressure on the existing network

resources owing to the exponentially increasing loads due to heavy IoT usage in daily life across myriad sectors.

We outlined the key characteristics that impact the performance of IoT applications, and have classified and kept into account the static part while increasing the network efficiency and broadening the scope of such applications. The logarithmic complexity of the proposed approach trumps the usual brute force solution to the deployment strategy problem, thereby making a contribution in improving the decision making process of deployment stage of IoT applications in fog computing environments.

Chapter 5

Distributed Decomposed Data Analytics in Fog enabled IoT Deployments

5.1 Introduction

The edge of the network plays a vital role in an IoT system, serving as an optimal site to perform operation on data before transmitting it over the network. One of the prime objectives is to generate useful information from the data in the IoT deployment. In the existing approaches for data analytics in IoT, all data from an IoT deployment is collected at a centralized location such as server(s) in data centre (i.e., cloud) and is then subjected to the desired data analytics model to generate information. Data in these IoT deployments moves from 'things' to cloud, and along this continuum passes through a number of network devices such as routers, gateways, etc. Each of these devices can be a potential candidate to host partial analytics capability to analyse the data, and further sending the calculated partial results instead of sending the raw data to cloud.

Contrary to the cloud which can be thought of as resource rich, the fog devices are resource constrained in nature whereby resource scaling (up/down and horizontal/vertical) cannot be done dynamically. An additional deployment of a complete data analytics computing module on the said resource might lead to full utilization of resources as the workload or data input increases, and may also affect its fundamental network operation. Hence, a careful placement of computing operations is sought for an efficient overall system performance, and thus, the approach of decomposed computing units seems ideal in an IoT environment with fog assistance. The intention here is to be able to learn a model directly either entirely in the fog environment or in a fog-cloud assisted environment based on local sensor data available to each fog node without sending all the raw data along the infrastructure pipeline.

For e.g., consider a case where the additional deployment of physical devices is not possible, and we are required to predict temperature based on given readings of limited environmental parameters such as humidity and air pressure for a large physical area. In such a scenario, sending all the raw sensing data to cloud for analyzing, and sending the result back to the user is neither feasible nor scalable [57]. This becomes even more challenging with limited Internet connectivity and growing volume of data. In such scenarios it is ideal to leverage fog computing architecture, where fog focuses on local knowledge in different local areas, while cloud can have a global view of the environment by combining all the information received from fog. These kinds of cases exists in industrial systems, such as monitoring complex hydraulic systems [188] to be working in order and to their desired capacity. Another related example can be to know the temperature of different areas of a large stadium during an event to get actionable insights for regulating air conditioning of those specific areas.

The contributions from this chapter have been summarized as follows:

- We present the fog based distributed data analytics solution for IoT deployments.
- The decomposition method used for distributing the analytics/intelligence in the infrastructure is based on Statistical Query Model and Summation Form, which makes it closed form in nature.
- To the best of our knowledge this is first attempt to decompose an analytics model (in closed form) to make it run in distributed manner in a fog based setting.
- The solution and methodology is generic is nature and is applicable to a wide variety of IoT based use case scenarios.
- The proposed approach has been applied to a real-world data set in a fog based testbed, and metrics related to resource utilization and quality of analytics solution have been presented.

The decomposition method used is not the contribution, but applying the decomposition method to the analytics model to run in a distributed manner in fog enabled IoT deployments is the contribution. What is novel is the decomposition made on a fog based distributed setting.

The chapter has been further structured as follows: §5.2 outlines the mathematical model of the system and the problem, and also presents the analytics model (multivariate linear regression) used in the work, §5.3 presents the decomposition methodology, §5.4 presents the experimental setup and data sets used, §5.5 presents results and discussion, §5.6 presents further discussion based on use-case, constraints and impact of changing data processing frequency, and finally §5.7 presents the conclusion and future work.

This work has been disseminated in the following publications **P8 - IEEE Access 2019** [14], **P6 - IEEE IoT Newsletter 2018** [13].

5.2 Mathematical Model of the System and Problem Formulation

The research question 3 (RQ3) is being addressed in this chapter, articulated as:

How to decompose data analytics computing programs to run between fog and cloud?

The objective of this research question was to develop a methodology for designing modular data analytics approach in IoT environments, while leveraging the fog computing paradigm. This is towards decomposing the data analytics programs, followed by the deployment of these decomposed units along the things-to-cloud continuum.

The remainder of this section gives the mathematical representation of the system (5.2.1) and presents the analytics model (multivariate linear regression) used to validate the approach (5.2.2).

5.2.1 Representation of System

We consider the network architecture with fog nodes forming a layer between IoT devices and the cloud. A graphical representation of the same was shown earlier in Fig. 2.4. For convenience, the same has been presented here as well in Fig. 5.1.

We examine scenarios where local IoT devices and the remote cloud services carry out data sensing, collection and analytics. With fog layer in the middle of IoT devices and cloud, the analytics computation can be distributed among fog nodes and can be collectively solved by either fog nodes alone or in a combined manner by fog nodes and cloud.

We consider a tree like network architecture in which an IoT device i is connected with its unique fog node j, which is further connected to the cloud. We have sensing devices that are transmitting their data to fog nodes, and they are sending their data towards cloud or another central location. The tree topology is a hierarchy of nodes with single root node at the highest level of hierarchy, which is connected to one or many nodes in the level below. The communication, computing and storage capabilities in node(s) increases as one moves from branches of the tree towards the root node. In this tree-like topology, the root represents cloud, intermediate nodes represent fog nodes and the leaf nodes represent IoT devices. Data in IoT deployments moves from things to cloud, or in

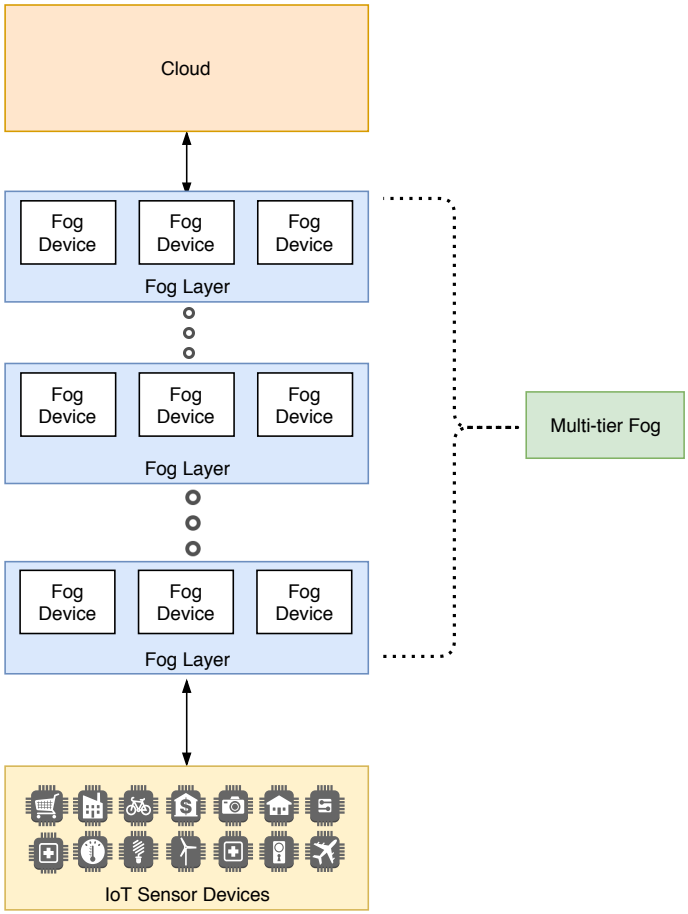


Fig. 5.1 Fog computing architecture representation as three tier IoT-fog-cloud architecture with multi-tier fog in a standardized IoT reference architecture model [41], [10], [12].

terms of tree representation from branches of the tree to cloud via fog nodes, allowing data to be processed closer to where its generated. The graphical representation of an end-to-end tree like network architecture was presented earlier in Fig. 2.6. For convenience, the same has been presented here as well in Fig. 5.2. Again, it should be noted that there can be topologies inside each layer/level, but the main structure and overall abstract topology is tree-like.

The neighbourhood of fog node j is denoted as $\mathcal{N}_j$ which is the set of IoT devices connected to it, and is written as $\mathcal{N}_j = \{n_j\}$ such that $i \in \mathcal{N}_j$. The range of communication of a fog node is defined in terms of its communication distance capability, i.e. any IoT device that can communicate with the fog node within its range can become a part of the fog node's neighbourhood.

We consider a discrete time domain $t \in \mathbb{T} = \{1, 2, ...\}$ such that an IoT device i at every time instance $t \in \mathbb{T}$, senses a d-dimensional vector $\mathbf{x}_t \in \mathbb{R}^d$ termed as sensed or context

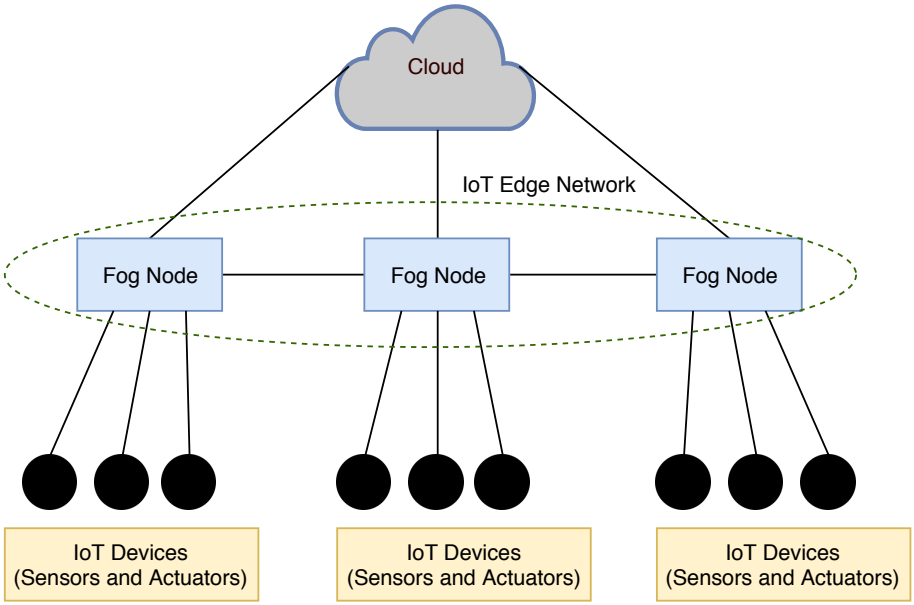


Fig. 5.2 Tree Topology in Fog Enabled IoT Deployments [14].

vector[1] containing contextual features/parameters for e.g. accelaration, g-force values, temperature, humidity, feature counter etc. The IoT device i communicates with its fog node j by sending sensed vectors at a a set frequency f. The transmission frequency f of IoT devices depends on a number of factors such as sampling rate and can range from seconds to minutes to hour(s) depending upon the use case. So at any time instance $t \in \mathbb{T}$, fog node j receives a set of sensed vectors from an IoT device, and overall sets of sensed vectors from its neighbourhood $\mathcal{N}_j$. A set frequency (or to say time-window or time-frame) is defined on fog node to perform the computing operation over data received till that point.

A frequency is also set on the IoT devices for sensing and transmission. Usually, in sensing (IoT) devices, a sliding window is specified using certain parameters that keep on appending the new sensed vectors and discarding the older ones based on their appearance without having the specific need to save them in local storage. Still, it is dependent on the IoT device(s) in use i.e., the sensing infrastructure, and varies from one to another.

5.2.2 Analytics Model - Multivariate Linear Regression

One of the most widely used and well-understood predictive analytics model is the multivariate linear regression [189], and thus we choose it as the analytics model in this work for the desired fog specific decomposition. Given a data set $\mathscr{D} = \langle \mathbf{x}_k^{in}, y_k^{out} \rangle_{k=1}^{m}$ with m training examples, where $\mathbf{x}_k^{in} \in \mathbb{R}^d$ and $y_k^{out} \in \mathbb{R}$ represents input-output pairs, the linear regression

[1]Note that any vector by defualt is assumed as column vector, the defualt convention is to write given vector as column vector.

estimates the current coefficient $\mathbf{w} \in \mathbb{R}^d$ which interprets the dependency between $\mathbf{x}_k^{in}$ and y_k^{out} :

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{k=1}^{m} \left(y_k^{out} - \mathbf{w}^{\mathbf{T}} \mathbf{x}_k^{in} \right)^2 \tag{5.1}$$

The predicted output by the linear regression model is $\hat{y}_k^{out} = \mathbf{w}^{\mathbf{T}} \mathbf{x}_k^{in}, k = \{1, 2...m\}$. The Root Mean Square Error (RMSE) over q predictions is defined as:

$$\varepsilon = \sqrt{\left(\frac{1}{q} \sum_{k=1}^{q} (y_k^{out} - \hat{y}_k^{out})^2 \right)} \tag{5.2}$$

Linear regression is also referred as Ordinary Least Squares (OLS) and Linear Least Squares (LLS). The methods to solve linear regression i.e., equation 5.1 are typically classified in two categories, Closed Form Solution or Direct Methods, and Iterative Methods. The same has been illustrated in Fig. 5.3.

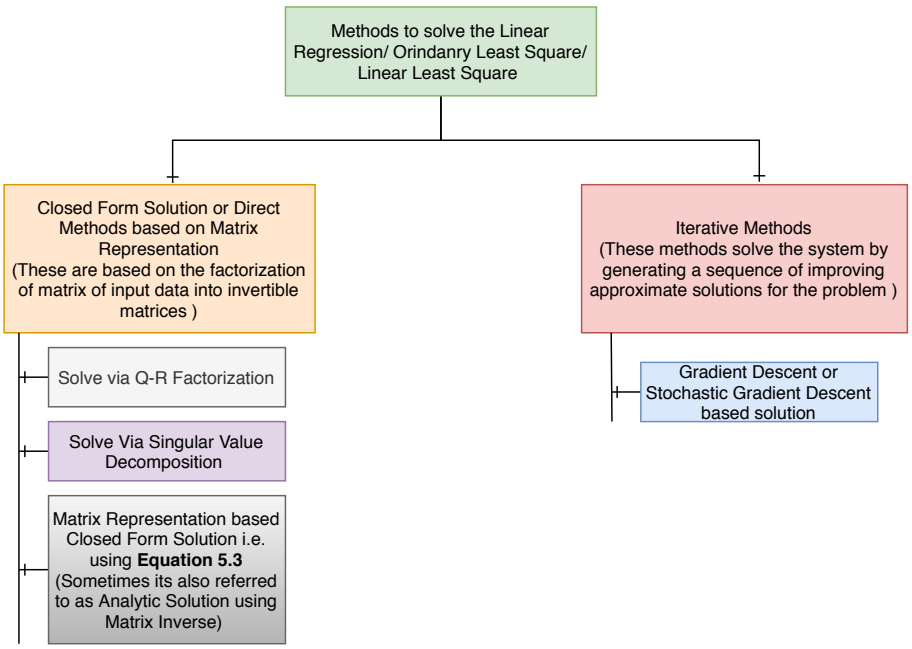


Fig. 5.3 Brief categorical representation of methods available to solve the problem of linear regression [14].

Each of these methods are equally used in different domains of study depending on number of factors including the type of problem being solved using linear regression, and the objective at hand such reducing time to get the solution, to get numerical stability [190] in the solution etc. The main difference in the available approaches is that each of them uses a different numerical method to solve the problem. There is extensive research in numeric linear algebra which look at and aim for parallelizing, and to certain extent

distributing these numerical operations [190], [191] with each having its own set objective. What is novel here is the utilization of fog computing paradigm to achieve that, and trying to get an efficient solution for the upcoming IoT and fog based applications and use case scenarios.

Objective: Our objective is to have the fog specific decomposition of the linear regression and focus on the arising trade-offs in latency (computing and communication), quality (analytical result obtained without decomposition and with decomposition) as the metrics for evaluation. We choose matrix representation based closed form solution for linear regression in this work and present its fog specific decomposition in the next section. The major difference and reason behind choosing closed form solution is that Gradient Descent and Stochastic Gradient Descent are iterative in nature and their convergence towards solution is time consuming, whereas matrix representation based closed form solution gives us a way for solving the least squares problem fit to the parameter without needing to use an iterative algorithm. Other reasons of preferring direct methods over iterative include their robustness and predictable behaviour in terms of resources required for their execution [192], [193]. The selection of the method depends on the context of the problem being solved [153]. So given m training examples : $(\mathbf{x}_1^{in}, y_1^{out}), (\mathbf{x}_2^{in}, y_2^{out}), ..., (\mathbf{x}_m^{in}, y_m^{out})$, we write a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ with $\mathbf{x}_1^{in}, \mathbf{x}_2^{in}, ..., \mathbf{x}_m^{in}$ as rows (each being a vector of dimension n), and column vector $y \in \mathbb{R}^{m \times 1}, y = [y_1^{out}, y_2^{out}, ..., y_m^{out}]^{m \times 1}$ as the targeted value corresponding to each row, then the solution for parameter $\mathbf{w}$ is:

$$\mathbf{w}^* = (\mathbf{X}^\mathbf{T}\mathbf{X})^{-1}\mathbf{X}^\mathbf{T}y \tag{5.3}$$

5.3 Decomposing Data Analytics Model

In this section we present the decomposition method for fog enabled IoT systems. We present the theoretical fundamentals that are used in this work for the decomposition of the multivariate linear regression model. The following mathematical notation is used in the sub-section 5.3.1 below:

- Data set $\mathscr{D} = \langle \mathbf{x}_l^{in}, y_l^{out} \rangle_{l=1}^m$
- X is input to the learning model — $X = \langle \mathbf{x}_l^{in} \rangle_{l=1}^m$
- Y is a function of X that we want to learn — $Y = F(\mathbf{x}) = \langle y_l^{out} \rangle_{l=1}^m$

5.3.1 Statistical Query Model and Summation Form

. The Statistical Query Model [194], [195] is often presented as a restriction on the Valiant model [196]. In Valiant model, the learning algorithm uses randomly drawn training

example $\langle \mathbf{x}_l^{in}, y_l^{out} \rangle$ to learn the target function f where as in Statistical Query model the learning algorithm uses some aggregates over the examples and not the individual examples. Given a function $f(\mathbf{x}, y)$ over instances (data points $\mathbf{x}$ and labels y), the statistical computing operation returns an estimate of the expectation of $f(\mathbf{x}, y)$(averaged over training/test distribution). Any learning model/algorithm that calculates sufficient statistics or gradients fits this model, and since these calculations may be batched, they are expressible as a sum over data points [162]. The class of such models/algorithms is large,[2] [162] for example Linear Regression, Naive Bayes, Logistic Regression, Support Vector Machine (SVM) are to name few among-st many others.

Authors in [162] show that any algorithm that fits the Statistical Query Model may be written in a certain "summation form". This form does not change the underlying algorithm and so is not an approximations, but is instead an exact implementation. They show that the summation form can be expressed in a map-reduce framework and the technique can achieve a linear speed up with the number of cores on a multicore machine. Their approach is a novel contribution to achieve parallelization for a large class of machine learning (ML) methods on a multicore machine. Their main objective was to develop a general and exact technique for parallel programming of a large class of ML algorithms for multicore processors.

The authors show that when an algorithm does sum over data then its computation can be distributed over multiple cores by dividing the data set into as many pieces as there are cores, give each core its share of data to sum the equations over, and aggregate the results at the end. They call this form of the algorithm as the "summation form". We use and extend their technique for the fog specific decomposition in our work to achieve distributed data analytics in fog enabled IoT deployments.

5.3.2 Summation Form of Linear Regression

In this section we present that how to put the algorithm into "summation form" as worked by authors in [162]. The solution for the parameter $\mathbf{w}$ as shown in equation 5.3 is:

$$\mathbf{w}^* = (\mathbf{X}^\mathbf{T}\mathbf{X})^{-1}\mathbf{X}^\mathbf{T}y \tag{5.4}$$

To put the above computation into summation form, it is reformulated into a two phase algorithm where first sufficient statistics are computed by summing over the data, and

[2] Although we present the fog specific decomposition only for Linear Regression, but in the same manner the methodology can be extended and modified for decomposition of other predictive analytics models as well.

those statistics are aggregated to get and solve:

$$\mathbf{w}^* = \mathbf{A}^{-1}\mathbf{b} \tag{5.5}$$

Where $\mathbf{A} = \mathbf{X}^\mathbf{T}\mathbf{X}$ and $\mathbf{b} = \mathbf{X}^\mathbf{T}y$, and is computed as follows:

$$\mathbf{A} = \sum_{\alpha=1}^{m} \left(\mathbf{x}_\alpha^{in}(\mathbf{x}_\alpha^{in})^\mathbf{T}\right) \quad \text{and} \quad \mathbf{b} = \sum_{\alpha=1}^{m} (\mathbf{x}_\alpha^{in} y_\alpha^{out})$$

m is the total number of training examples in the dataset. The computation of $\mathbf{A}$ and $\mathbf{b}$ can now be divided into equal size pieces and distributed amongst the cores as presented by authors [162] to achieve parallelization for ML programs on multicore machines. In terms of their proposed Map-reduce framework for ML the above computation is divided among mappers. As in this case, one set of mappers is used to compute $\sum_{subgroup}\left(\mathbf{x}_\alpha^{in}(\mathbf{x}_\alpha^{in})^\mathbf{T}\right)$ and another set to compute $\sum_{subgroup}(\mathbf{x}_\alpha^{in} y_\alpha^{out})$. A set of reducers then respectively sum up the partial values for $\mathbf{A}$ and $\mathbf{b}$, and finally computes the solution $\mathbf{w}^* = \mathbf{A}^{-1}\mathbf{b}$ (i.e. equation 5.5).

In our work instead of distributing the computing operations over cores, the computing operation of summation is put on each fog node in the infrastructure. So each of the fog node performs the computing operation of summing over the collected data and then sending the summarized outputs to either fog node or cloud (depending on the implementation and use-case) rather than sending raw data over the network.

In many industrial settings and IoT deployments, the data is collected and stored in a decentralized manner. When the data generation/ storage is itself distributed, then it appears more desirable to also process/analyse it in a distributed fashion to avoid the bottleneck of data transfer to the centralized cloud.

The pseudo code of the implementation discussed in this section and as used in the experiments has been presented in Algorithm 5.1 and Algorithm 5.2. Algorithm 5.1 represents the linear regression component running on fog nodes and Algorithm 5.2 represents the component running in cloud for the fog based distributed analytics approach. In cloud centric approach, both Algorithm 5.1 and 5.2 run on the cloud as the whole processing happens there and fog node acts as normal gateways without the analytics component running on them.

5.4 Experimental Setup, Data-sets and Validation

The experiment was performed on the OpenStack VM (Virtual Machine) instances. The setup consists of a total of 5 VMs as shown in Fig. 5.4. The configuration details of the VMs are presented in table 5.1. All VMs have Ubuntu 18.04.2 LTS as their operating system and Intel Xeon processors (@2.60 GHz).

Algorithm 5.1 FLRC (Fog Linear Regression Component) - Decomposed Computing Program Running on each Fog Node VM

Initialize : $\mathbf{A_t} = 0$, $\mathbf{b_t}$ [ ] = 0
▷ $\mathbf{A_t}$ will be a real value and $\mathbf{b_t}$ will be a vector with same dimension as $[\mathbf{x}_\alpha^{in}]_t$. $\mathbf{A_t}$ and $\mathbf{b_t}$ will contain the partial calculated values after the execution of the program.
Input : X_t [ ] = $[\mathbf{x}_\alpha^{in}]_t$, Y_t [ ] = $[y_\alpha^{out}]_t$
▷ $[\mathbf{x}_\alpha^{in}]_t$ and $[y_\alpha^{out}]_t$ represent the data received in set processing frequency t
Output : Processed outputs calculated in the set processing frequency

```
function FLRC( X_t [ ], Y_t [ ])
    for index = 0 to (size of X_t [ ]) - 1 do
        A_t + = DOTPRODUCT (X_t [index], TRANSPOSE (X_t [index]))
        b_t [ ] + = X_t [index] * Y_t[index]
        index + = 1
    end for
    return (A_t, b_t [ ])
end function
```

Algorithm 5.2 CLRC (Cloud Linear Regression Component) - Program Running on Cloud to combine the partial results obtained from Fog Nodes

Initialize : $\mathbf{A} = 0$, $\mathbf{b}$ [ ] = 0, $\mathbf{w}^*$ [ ] = 0
Input : Processed outputs obtained from Fog Nodes i.e. Different $\mathbf{A_t}$'s and $\mathbf{b_t}$'s [ ] received during the whole duration of the experiment
Output : Regression Coefficients i.e. Linear Regression Model in the Distributed Approach

```
function CLRC(A_t's , b_t's [ ])
    A = SUM of all A_t's received
    b [ ] = SUM of all b_t's received
    w* [ ] = b / A
    return (w* [ ])
end function
```

Table 5.1 Experimental Configurations

Use and type of VM	VCPUs	RAM (in GB)	DISK (in GB)	Number of VMs
VM used for streaming data	2	4	10	1
VMs acting as Fog Nodes	4	8	20	3
VM acting as Cloud	8	16	30	1

The configuration of VMs acting as fog nodes were kept inline to the commercially available IoT Gateways by Dell and Intel [197], [198]. The computing capacity of the VMs

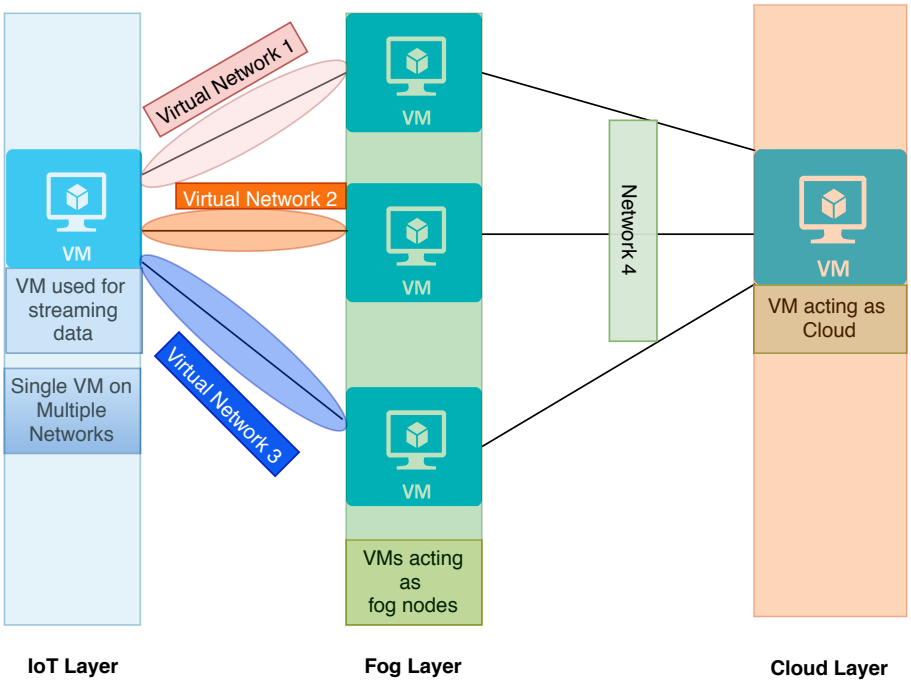


Fig. 5.4 Experimental setup deployed on OpenStack.

increases as we move up in the hierarchy. We used real-world dataset from UCI repository [199] in our experiments to evaluate the performance of the proposed mechanism. The dataset [200] contains 9358 instances of hourly averaged responses of chemical compounds and environmental parameters from an air quality sensor. In the dataset, the parameters used to measure the air pollution of a specific area include — CO (ground truth values), PT08.S1 (CO), NMHC (Non Metanic HydroCarbons, ground truth values), Benzene (C6H6, ground truth values), PT08.S2 (NMHC), NOx (ground truth values), PT08.S3 (NOx), NO2 (ground truth values), PT08.S4 (NO2), PT08.S5 (O3), temperature, relative humidity, absolute humidity. These parameters are used to measure the air pollution of a specific area. It consists of a total of 13 features out of which 5 represent the ground truth values for the same type. The authors in [200] have used this dataset for benzene estimation in an urban environment pollution scenario. We also use this dataset for benzene estimation in our linear regression task. We remove the ground truth values from the dataset and thus remain with 8 features as input ($\mathbf{x}_i^{in}$) to the regression task with Benzene (C6H6, ground truth values) as the targeted variable y_i^{out}.

The data is column-standardized (mean centering and scaling) and normalized i.e. each vector $\mathbf{x}_i^{in}$ is mapped to $\frac{\mathbf{x}_i^{in}-\mu}{\sigma}$ with mean value μ and variance σ^2, and scaled in $[0,1]$.

The 8 features correspond to 8 sensors in the real-word setting. The data split is made as 70-30 i.e. 70% (approximately 6540 instances) data is used for model training and 30% (approximately 2818 instances) is used for testing. We then randomly divide

the train data into three equal parts each containing approximately 2180 instances and each part is streamed on row by row basis to corresponding VM acting as fog node gateway. This corresponds to a real-world setup where a group of 8 sensors is present in 3 different locations (so 24 sensors in all) which are streaming their sensed values to their corresponding gateways. The graphical representation of such an equivalent real-world setup is as shown in Fig. 5.5

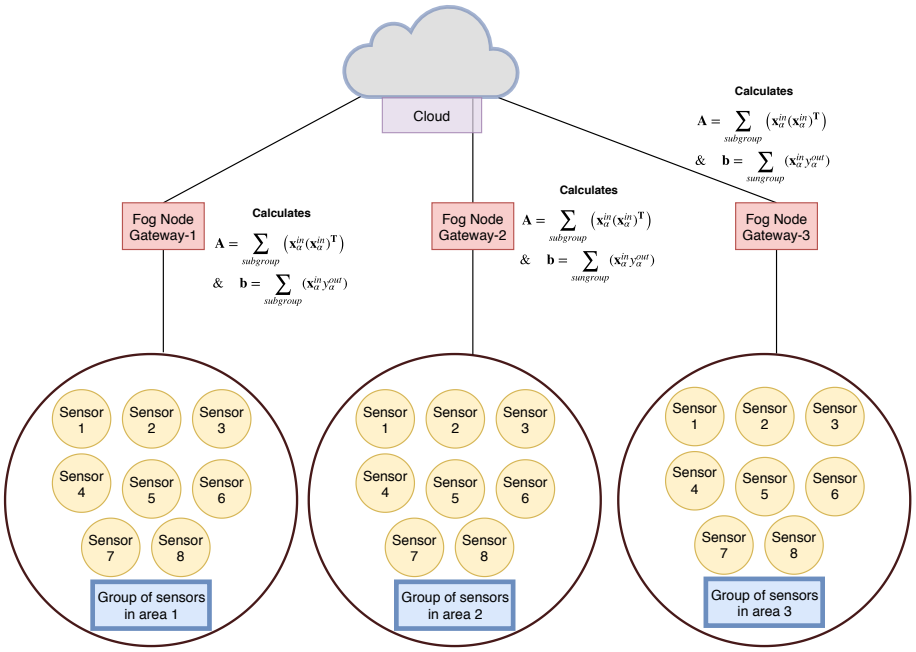


Fig. 5.5 Real-world equivalent representation of the experimental setup deployed on OpenStack [14].

We selected MQTT [165] as the streaming protocoal in our setup. The MQTT architecture comprises of two components, namely MQTT clients (such as publishers and subscribers) and MQTT broker (for mediating messages between publishers and subscribers). In our setup these components are as follows:

- **MQTT Publisher:** Script running on streaming VM acts as the MQTT publisher client. It streams the data on row by row basis every second (i.e 1 row per second) to each of three VMs acting as fog node gateway.
- **MQTT Broker:** The VMs acting as fog node gateway act as broker between VM acting as cloud and VM streaming data as sensor. The VMs acting as fog node gateway subscribe to the streaming VM acting as publisher.
- **MQTT Subscriber:** Another script running on VM acting as cloud subscribes to the VMs acting as fog node gateways.

The processing frequency for data received at fog node VMs (i.e. MQTT broker) was set to 5 seconds. We used Paho [201] as MQTT client library and Mosquitto [202] as MQTT broker library in our implementation. The VMs acting as fog node gateway receive data, performs the computing operation as described earlier and presented in Algorithm 4.1, and sends the output to the VM acting as cloud.

The experiment was performed in two scenarios, one where the VMs acting fog node gateways perform the computing operation of calculating subgroups of **A** and **b** as per Algorithm 5.1, and send the processed output to the cloud VM. Second scenario is the traditional centralized setup where fog gateway VMs receive the streaming data and forward it as is to the cloud, and the whole data analytics operation takes place in the cloud VM. The results from the first scenario have been labelled as *Distributed Approach* and from second scenario as *Cloud Centric Approach*.

5.5 Results and Discussion

The system utilization metrics such as CPU, memory and bandwidth utilization were noted in both the scenarios and have been presented in this section. Along with that, metrics to measure the quality of analytics solution in both scenarios were evaluated and have also been presented here. The centralized solution acts as a comparative measure against the distributed approach.

The following notations have been used in this section:

- **Fog Layer**: Average value of metrics obtained from the 3 VMs acting as fog nodes
- **Cloud Layer**: VM acting as Cloud

The experiment runs for approximately around 35 minutes. We used Python Resmon [203] and Glances [204] to measure the resource utilization metrics of VMs in the experiment.

5.5.1 Accuracy and Distribution Plots

The Linear Regression model generated in both the approaches was tested on the same test data, and results of the same have been presented in table 5.2. The values presented for both RMSE (Root Mean Square Error) and variance score/accuracy are up-to 4 significant digits after the decimal.

The lower the RMSE the better the model. The results suggest that both the approaches generate the same model, as the RMSE and variance score values obtained are exactly the same. So distributed approach can be used to obtain the same results as one would have obtained from the traditional centralized approach.

Table 5.2 Accuracy of Generated Models

Approach	Root Mean Square Error (RMSE)	Variance Score/ Accuracy
Distributed Approach	0.0384	0.9673
Cloud Centric Approach (Summation Form)	0.0384	0.9673

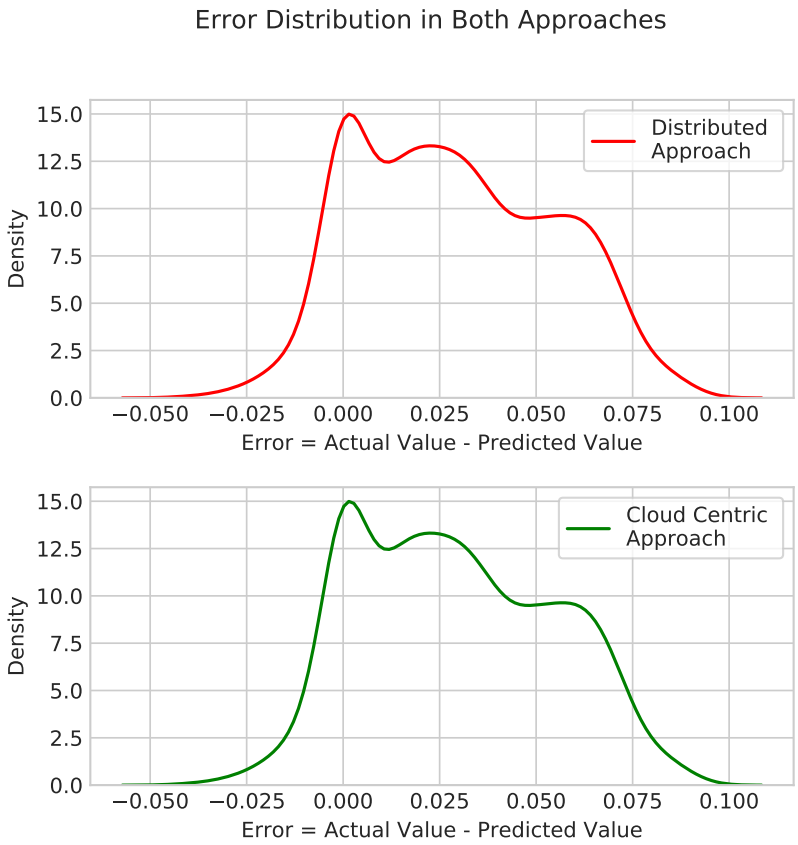
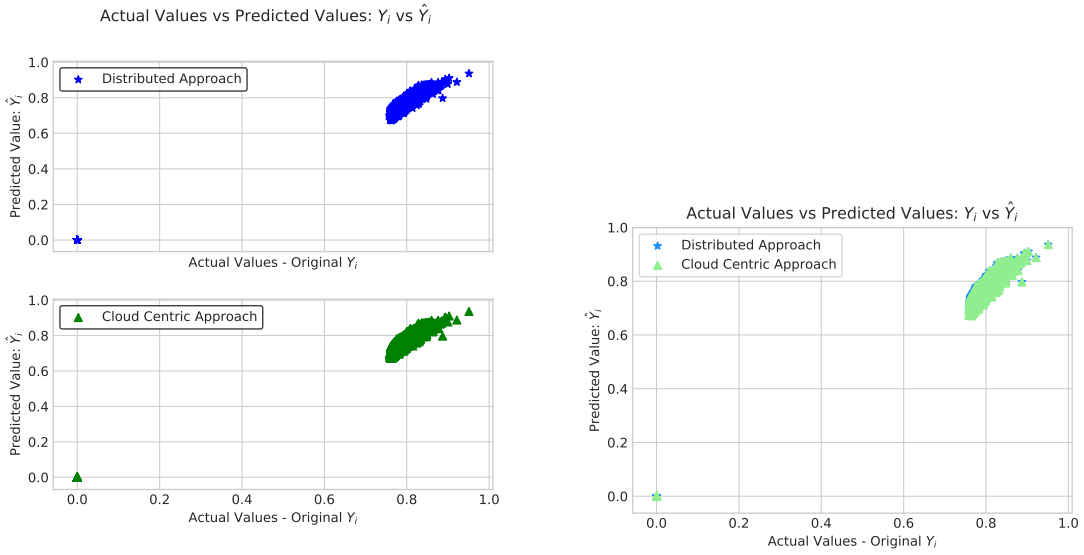


Fig. 5.6 Error distribution plot in both approaches.

The error distribution in both the approaches has been presented in fig. 5.6. The X-axis here represents $\Delta Y = Y_{test} - Y_{predicted}$ and Y-axis represents the probability density of it. The first thing to notice here is that the errors are centered at zero in both approaches i.e. the most often found error is zero, which is good. The distribution plot from both the approaches look exactly the same, which suggests that distributed approach can be used in fog enabled IoT deployments. In both approaches the errors are more on the positive side as we can see from the heights of the distribution on the +ve side and there are fewer error on the -ve side. The errors are fairly small in both approaches which suggests that its a fairly decent solution.



(a) Scatter Plot of actual and predicted values in both approaches

(b) Scatter Plot of actual and predicted values in both approaches plotted together

Fig. 5.7 Scatter plot that shows that the distributed and cloud centric approach both trace out each other.

Fig. 5.7 presents the scatter plot of actual and predicted values in both the approaches. Fig. 5.7a represents the plot on shared axis and fig. 5.7b represents the plot on same axis. As visible from the plot, the distributed approach and cloud centric approach trace out each other.

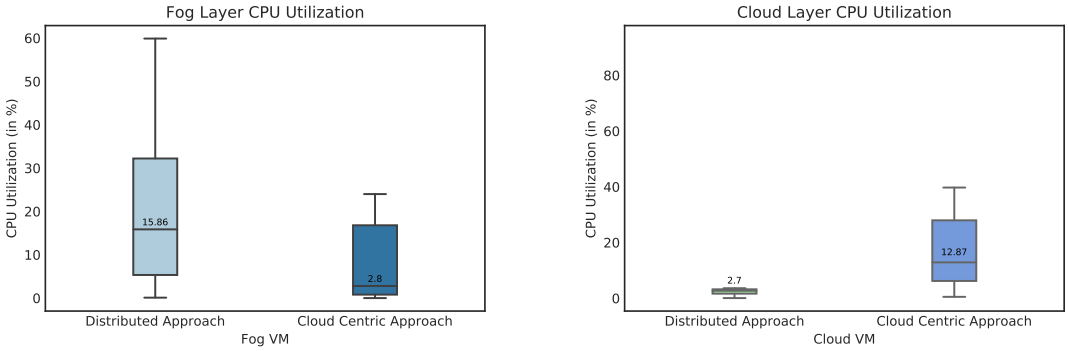
5.5.2 CPU Utilization

The CPU utilization of various entities involved in the experimental setup has been presented as box plots in Fig. 5.8. The values shown in the figure represents the median values. The results of CPU utilization have been discussed below:

Fog layer CPU utilization: CPU utilization of fog node VMs increases in the distributed approach as now they are also performing the analytics operation rather than just forwarding the data to cloud for analysis. This also adds to efficient resource utilization of these devices.

Cloud layer CPU utilization: CPU utilization of cloud VM is less in distributed approach and more in the traditional cloud-centric approach. The reason for this is that in cloud centric approach all the processing happens in cloud, while in distributed approach cloud only sums up the partial results obtained from fog nodes, thus leading to less CPU utilization in the latter.

This is beneficial for the user as now the monetary cost will be less for the cloud service utilization under the ‘pay as you go’ model. In most IoT based deployments the gateways



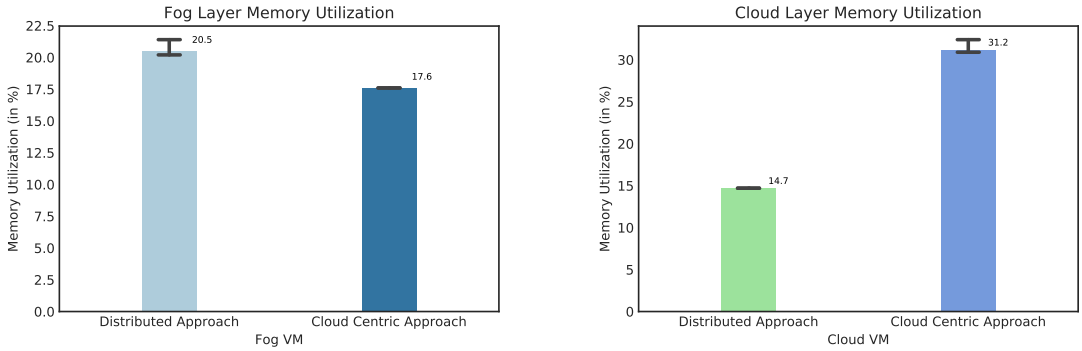
(a) Fog Layer CPU Utilization (b) Cloud Layer CPU Utilization

Fig. 5.8 CPU utilization visualized for both the distributed and cloud centric approach.

are usually owned by the user so effectively the user will have to pay significantly less amount in the distributed approach.

5.5.3 Memory Utilization

The memory utilization has been presented as bar plots in Fig. 5.9. The values shown in the plot represent the median values. This follows the same behavior as for CPU utilization as discussed above.



(a) Fog Layer Memory Utilization (b) Cloud Layer Memory Utilization

Fig. 5.9 Memory utilization visualized for both the distributed and cloud centric approach.

5.5.4 Data Reduction

The bar plot in Fig. 5.10 represents the reduction in amount of data being streamed from fog VMs to cloud VMs in both approaches. There is 80% reduction in the amount of data being streamed in distributed approach as compared to the cloud centric approach. The

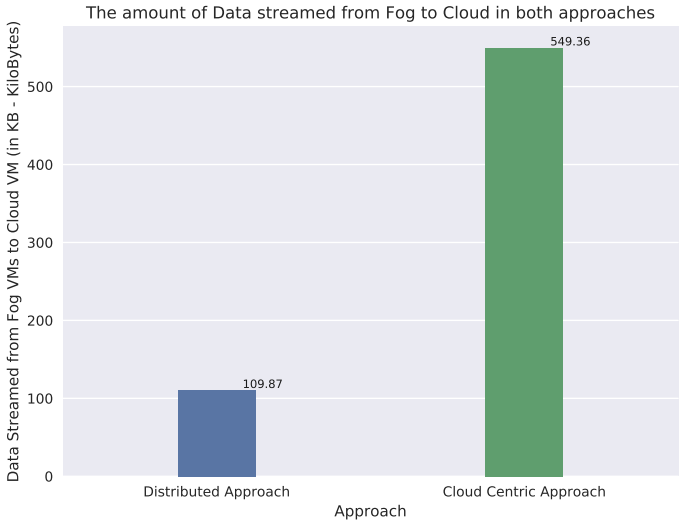


Fig. 5.10 Reduction in amount of data being streamed from fog VMs to cloud VM in fog based distributed approach.

reason for this reduction is the Fog Linear Regression Component (Algorithm 5.1) running on fog VMs in distributed analytics approach while in cloud centric approach they just act as normal gateways to forward the data received to cloud for analytics and the whole analytics operation happens there.

5.5.5 Time

We also measured the time required to calculate the regression coefficients in both the approaches i.e. to generate the final linear regression model. In both the approaches, the final model is calculated at the cloud. The values have been presented in table 5.3.

Table 5.3 Time taken to calculate regression coefficients in both approaches.

	Distributed Approach	**Cloud Centric Approach**
Time Taken (in milliseconds)	2.87 ms	138.62 ms

There is significant reduction in the amount of time required to calculate the final model in distributed approach compared to the cloud centric approach. The reason behind this is that in distributed approach cloud sums (Algorithm 5.2) the partial outputs obtained from fog nodes and then calculates the final model, while in cloud centric approach, the entire end-to-end processing happens in cloud (i.e. both Algorithm 5.1 and Algorithm 5.2 run on cloud).

5.6 Further Discussion

5.6.1 Use Case and Constraints

Given that the streaming rate of data from sensor/base node to the fog device is fixed, there are two kinds of use cases that define the role of the fog node in the distributed computing approach:

1. The first use case is where the fog node uses its resources for pre-processing the data acquired from sensors. This serves two purposes:
   (a) **Data reduction:** Lesser data is sent from fog node to the cloud
   (b) **Decreased computation time (in the cloud):** Since the data is already pre-processed with some initial operations, the time taken for the cloud component to process the entire data set into the desired output is lesser than the regular cloud-centric case. This is in addition to no overheads in terms of the total time required in the complete end-to-end process.
2. The second use case is where the fog node acts as both the data processing and decision making entity. This is particularly for latency sensitive use cases, where the data streaming and processing scenarios require latency critical decision making.

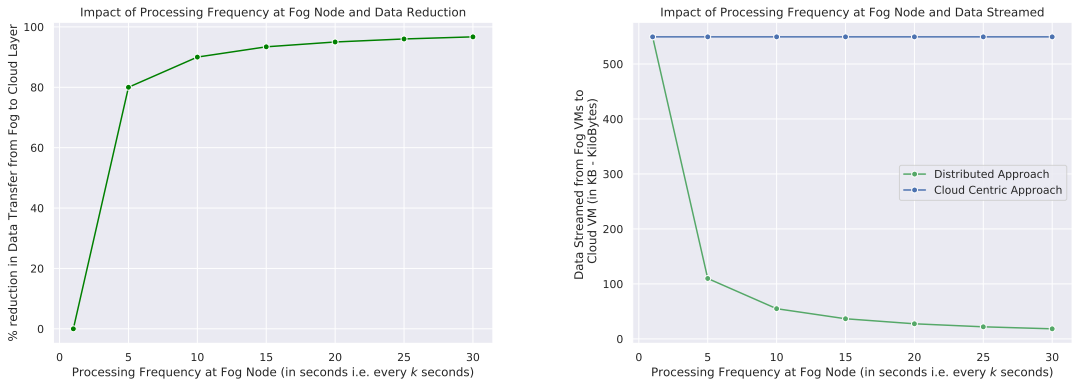
The three constraints that drive the above scenarios include:

1. Rate of streaming of data from end node to fog layer
2. Resource capacity of fog layer
3. Nature of use case:
   (a) latency sensitive / time critical decision making
   (b) regular computation required from data set

5.6.2 Impact of changing processing frequency at Fog Node and Data Reduction

With the above understanding we also measured the effect of varying processing frequency for the data received at fog node and corresponding reduction in data transfer from Fog Layer to Cloud Layer. The corresponding plots has been presented in Fig. 5.11a and 5.11b.

As visible from the plot, if we keep the data processing frequency at fog nodes same as the data receiving rate then there is no reduction in the data transfer, but as we gradually increase the processing frequency the data reduction becomes significant. The decrease in



(a) Effect of changing data processing frequency at fog node VMs and corresponding % reduction in data transfer to cloud using fog based distributed data analytics approach compared to traditional cloud centric approach

(b) Effect of changing data processing frequency at fog node VMs and corresponding reduction in data transfer to cloud using fog based distributed data analytics approach compared to traditional cloud centric approach.

Fig. 5.11 Impact of changing data processing frequency at fog node and data reduction.

data reduction with increasing processing frequency almost saturates after some point, and % reduction in data transfer becomes smaller.

In cloud centric approach even if we increase the data processing frequency at fog nodes, they still send all the received data, just that now they send it collectively, while in distributed approach, only the processed data is sent further to cloud and hence gain in data reduction. With higher processing frequency at fog node, which in turn means bigger buffer size to process the received data, the approach in turn effectively becomes centralized in nature, and thus the gain in reduction does not increase much after a certain point.

5.7 Summary and Conclusions

The aim of research question 3 (RQ3) was to develop a methodology for designing modular data analytics approach. In this chapter, we presented the methodology, approach and results for adopting distributed decomposed data analytics in fog enabled IoT deployments. The benefit of using fog computing for all IoT based applications may not be obvious since benefits gained may not be significant to motivate the use of the edge of the network. It might also be argued that it is more desirable to develop cloud centric solutions with sufficiently large number of resources available on hand, rather than designing fully distributed computing programs/algorithms which might bring along additional complexities. However, the number of data centres is less likely to grow at the same rate as the number

of devices at the network edge (and thus the generated amount of data) being connected to the Internet, since traditional data centres consume a lot of power and global network bandwidth, and have begun to raise the impending concern of an increased carbon footprint.

Overall, keeping in mind the challenges, the decomposition of analytics programs in fog assisted IoT environments does look promising towards the effort to design efficient distributed data analytics solutions and making the edge of network smarter, and in line with the vision of distributed computing towards future networks.

Chapter 6

Conclusions and Future Work

This chapter summarizes our work presented in the dissertation, and maps the direction for future work. The main objective of this dissertation was to examine and develop approaches on providing fog computing support for IoT applications. This includes prototyping, validating, and building approaches that can improve the decision making process in the life cycle of IoT applications towards leveraging the fog computing paradigm. Fig. 6.1 gives a quick overview of the different areas studied as part of this work as presented in different chapters, and the corresponding contributions made in those areas, thereby summarising the dissertation as a high-level graphical representation. The dissertation highlighted the benefits that can be brought with using a fog computing assisted approach in the life cycle of IoT applications, which was formalized in the research hypothesis with three research questions presented in Chapter 1. It focused on the design, development and deployment stages of IoT application life-cycle, and on the data analytics functionality of these IoT applications in the fog computing environment.

The previous chapters of the dissertation have addressed each of the research questions in detail, providing insights into the foundational research work while mapping contributions to the core areas of IoT and fog computing. This chapter focuses on concluding the work presented, and a discussion on possible avenues for further research. It provides a short summary (§6.1) of the chapters followed by presenting the directions for future work (§6.2).

6.1 Summary

Chapter 3 presented a design and development methodology for IoT applications leveraging the fog computing paradigm in a smart dairy farming setup for animal behaviour analysis and health monitoring. It included the design principles and changes required in the IoT application life cycle stages in order to have an application architecture that complements

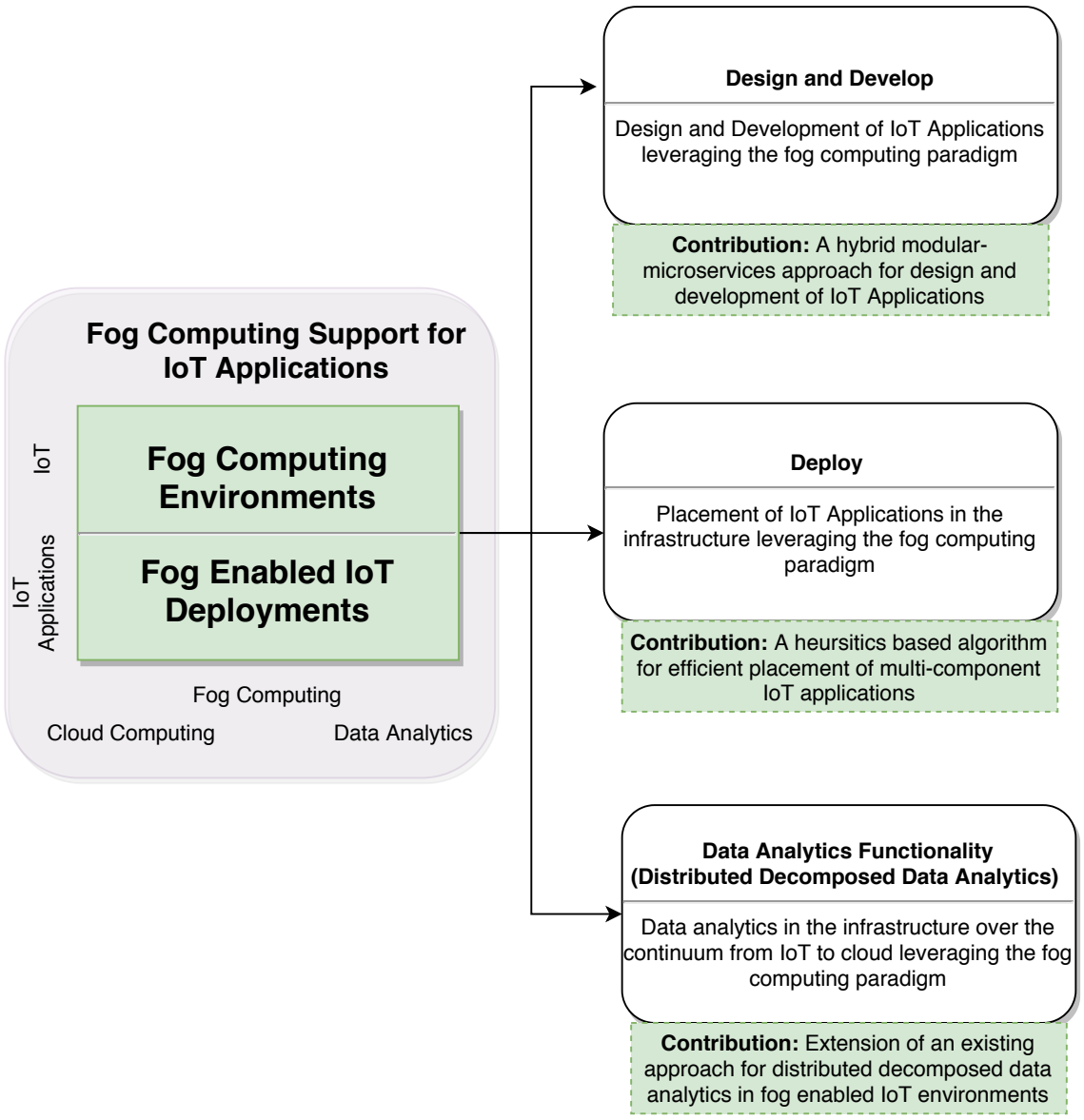


Fig. 6.1 A modular representation of subject-fields and contributions made in through the research work as presented in this dissertation.

the fog computing paradigm. It investigated the benefits of using a fog computing assisted approach in scenarios with low/scarce Internet connectivity, such as in remote geographical locations of farms. It presented the benefits of having a distributed modular application architecture using microservices coupled with the fog computing paradigm, and measured its subsequent benefits in terms of data reduction, efficient computing resource usage, scalability, and fault tolerance. It also presented the computing resource benchmarking of the developed solution. Early lameness detection being the objective of the IoT solution, it also presented the associated concepts, methodology used from the data analytics domain to achieve that.

Chapter 4 presented a fog-cloud placement approach (Module Mapping Algorithm) to place distributed applications in a fog-cloud environment. It presents how to efficiently deploy multi-component IoT applications in the infrastructure, leveraging fog computing. The proposed approach ensures efficient resource utilization in the infrastructure. It sorts both the nodes and application modules according to the available capacity and requirements, and maps the modules when the constraint is satisfied. The proposed approach reflects the way to reduce resource under-utilization for distributed IoT applications. It also highlights how fog-cloud inter-operation can minimize end-to-end latency compared to the cloud-based approaches.

Chapter 5 presented the distributed decomposed data analytics in fog enabled IoT deployments. It showed how the traditional data analytics algorithms can further be modularized to run on resource constrained devices in fog computing environments. It presented the methodology used for decomposition, and validated the proposed approach in a fog computing test-bed environment. The benefits arising from having distributed decomposed data analytics were noted and measured in terms of efficient computation resource usage, reduction in amount of data, and reduction in training time of the model. It also presented the discussion on use-case constraints and the effect of changing the data processing frequency at the fog layer of the setup.

6.2 Future Work

A discussion on the possible avenues and extension for further research is presented here, along with the limitations and shortcomings.

• **CHAPTER 3:** In the SmartHerd project work presented in Chapter 3, one of the limitations is a plausible failure situation of the fog node. In the current setup there was only one fog node used in the deployment, so if that fog node fails, the system will face a downtime. In the likely event that this fog node (SmartHerd gateway) is overloaded or is unavailable, the sensors have a data retention capacity of up to 12 hours. Moreover, the gateway also has a local database which stores the data before pre-processing, and the MQTT publisher saves the state of the last successfully transmitted events. Thus, the system would then continue from where it stopped in case of a network outage. However, a natural extension to the work would be to further increase the fault tolerance and resilience of the system by introducing the co-existence of multiple gateways in the infrastructure which are synchronized with each other, and data can be redirected between them in-case of failure of one gateway.

Another direction would be to further analyze and improve the fault tolerance and resilience for the developed system (i.e., distributed modular application architecture using

microservices in fog computing environments) using the concepts from Chaos Theory [180], [181]. Appendix A presents a complementary extension to this direction of work. It presents the initial empirical analysis to this end, and serves as a guiding point for further enhancements that can be done. Another possible direction would be to look at the further granularity of the computational resource benchmarking of the system i.e., instead of quantifying the minimum resources as number of CPUs, quantify it as CPU cycles etc.

From the machine learning perspective, further work may include a move towards ensemble learning and federated learning methodology. The deployment under the follow-on project MELD on multiple farms presents an opportunity to look into these approaches in such a setup.

• **CHAPTER 4:** One of the natural extensions of the work would be to look into the dynamic nature of the DAG of the application. Since the time the application deployment framework presented in Chapter 4 of the dissertation was disseminated, to the time of writing the dissertation, there has been new work in the area that has addressed some of the shortcomings of our work. We have included that activity below as complementary addition to the dissertation. All the work included below addresses certain aspects of the dynamic nature of the DAG.

Authors in [205] further refined the edge-ward algorithm of iFogSim [62] with a posteriori management policy to guarantee the application deadlines and optimize fog resource exploitation.

Authors in [206] proposed a software platform that provides a mapping strategy for IoT application placement in fog computing environments, that optimizes resource utilization, bandwidth and response time constraints. Their approach permits to choose among a cloud-only, a fog-only or a cloud-to-fog deployment policy.

Authors in [207] proposed a QoS-aware deployment strategy for multi-component IoT applications in fog infrastructure. It determines eligible deployment of a multi-component application by pre-processing plus backtracking. It utilizes a greedy backtracking algorithm as the heuristic that uses system requirements of the application as constraints.

Authors in [187] exploit Monte Carlo simulations to handle the communication links' variations, and to perform pre-processing plus backtracking to determine the final eligible deployment of a multi-component IoT application.

Authors in [186] presented an exhaustive survey of the existing methodologies to solve the application placement problem in fog infrastructure.

Authors in [127] recently present a detailed survey of service placement problem in fog computing environments.

Another direction of work would be to have a combined deployment and migration framework in fog computing environments, where both the deployment and migration of application components takes place during run time depending on the changes in the environment.

• **CHAPTER 5:** One of the possible directions for future work in the distributed decomposed data analytics would be to develop an approach where the decomposition itself is automated and happens dynamically during run-time. Another direction would be to have a dynamic service migration frame in place, which allows automated migration of the data analytics functionality between fog and cloud. It should also be mentioned here that from the software development perspective, having data in one place is useful, as it allows processing as and when required. This makes it easier to perform: (i) parameter tuning, (ii) retraining, (iii) understanding when more data could be needed. There needs to be further investigation to match the software and machine learning development practices, while incorporating the edge of the network for data analytics. It becomes important to investigate in this direction of research given that the number of data centres is less likely to grow at the same rate as the number of devices at the network edge (and thus the generated amount of data) being connected to the Internet, since traditional data centres consume a lot of power and global network bandwidth, and have begun to raise the impending concern of increased carbon footprint.

Fog Computing as a Fluid Computational Platform: With server-less computing in picture, the idea here is that the whole infrastructure acts as a computation platform. It can be seen as a system consisting of various computing resources along the technology path from sensors to cloud. This is a future vision, where there is no hard boundary between edge or fog and cloud, and the whole infrastructure is visualized as a fluid computational resource. The idea is that one can perform and move computation anywhere in this complete technology path, and can gain benefits from execution of services closer to the data sources or consumer depending on the use case.

Bibliography

[1] J. P. Dias and H. S. Ferreira, "State of the software development life-cycle for the internet-of-things," *CoRR*, vol. abs/1811.04159, 2018. arXiv: 1811.04159. [Online]. Available: http://arxiv.org/abs/1811.04159.

[2] L. F. Rahman, T. Ozcelebi, and J. Lukkien, "Understanding iot systems: A life cycle approach," *Procedia Computer Science*, vol. 130, pp. 1057–1062, 2018, The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.04.148. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050918305106.

[3] H. Cao, M. Wachowicz, C. Renso, and E. Carlini, "Analytics everywhere: Generating insights from the internet of things," *IEEE Access*, vol. 7, pp. 71 749–71 769, 2019.

[4] N. B. Ruparelia, "Software development lifecycle models," *SIGSOFT Softw. Eng. Notes*, vol. 35, no. 3, pp. 8–13, May 2010, ISSN: 0163-5948. DOI: 10.1145/1764810.1764814. [Online]. Available: https://doi.org/10.1145/1764810.1764814.

[5] M. Taneja, J. Byabazaire, A. Davy, and C. Olariu, "Fog assisted application support for animal behaviour analysis and health monitoring in dairy farming," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Feb. 2018, pp. 819–824. DOI: 10.1109/WF-IoT.2018.8355141.

[6] J. Byabazaire, C. Olariu, M. Taneja, and A. Davy, "Lameness detection as a service: Application of machine learning to an internet of cattle," in *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan. 2019, pp. 1–6. DOI: 10.1109/CCNC.2019.8651681.

[7] M. Taneja, N. Jalodia, J. Byabazaire, A. Davy, and C. Olariu, "Smartherd management: A microservices-based fog computing–assisted iot platform towards data-driven smart dairy farming," *Software: Practice and Experience*, vol. 49, no. 7, pp. 1055–1078, 2019. DOI: 10.1002/spe.2704. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2704. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2704.

[8] M. Taneja, N. Jalodia, P. Malone, J. Byabazaire, A. Davy, and C. Olariu, "Connected cows: Utilizing fog and cloud analytics toward data-driven decisions for smart dairy farming," *IEEE Internet of Things Magazine*, vol. 2, no. 4, pp. 32–37, Dec. 2019, ISSN: 2576-3199. DOI: 10.1109/IOTM.0001.1900045.

[9] M. Taneja, J. Byabazaire, N. Jalodia, A. Davy, C. Olariu, and P. Malone, "Machine learning based fog computing assisted data-driven approach for early lameness detection in dairy cattle," *Computers and Electronics in Agriculture*, vol. 171, p. 105 286, 2020, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2020.105286. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016816991931840X.

[10] M. Taneja and A. Davy, "Resource aware placement of data analytics platform in fog computing," *Procedia Computer Science*, vol. 97, pp. 153–156, 2016, 2nd International Conference on Cloud Forward: From Distributed to Complete Computing, ISSN: 1877-0509. DOI: http://dx.doi.org/10.1016/j.procs.2016.08.295. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050916321111.

[11] M. Taneja and A. Davy, "Poster abstract: Resource aware placement of data stream analytics operators on fog infrastructure for internet of things applications," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct. 2016, pp. 113–114. DOI: 10.1109/SEC.2016.44.

[12] M. Taneja and A. Davy, "Resource aware placement of iot application modules in fog-cloud computing paradigm," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 1222–1228. DOI: 10.23919/INM.2017.7987464.

[13] M. Taneja, N. Jalodia, and A. Davy, *Towards Decomposed Data Analytics in Fog Enabled IoT Deployments - IEEE Internet of Things Newsletter*, https://iot.ieee.org/newsletter/september-2018/towards-decomposed-data-analytics-in-fog-enabled-iot-deployments.html, Sep. 2018.

[14] M. Taneja, N. Jalodia, and A. Davy, "Distributed decomposed data analytics in fog enabled iot deployments," *IEEE Access*, vol. 7, pp. 40 969–40 981, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2907808.

[15] IEEE, *Define iot - ieee internet of things*, https://iot.ieee.org/definition.html, (Accessed on 05/14/2018).

[16] *What is internet of things (iot)? - definition from whatis.com*, https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT, (Accessed on 05/14/2018).

[17] A. Whitmore, A. Agarwal, and L. Da Xu, "The internet of things—a survey of topics and trends," *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, Apr. 2015, ISSN: 1572-9419. DOI: 10.1007/s10796-014-9489-2. [Online]. Available: https://doi.org/10.1007/s10796-014-9489-2.

[18] 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, Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications – Big Data, Scalable Analytics, and Beyond, ISSN: 0167-739X. DOI: http://dx.doi.org/10.1016/j.future.2013.01.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000241.

[19] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM Comput. Surv.*, vol. 50, no. 3, 32:1–32:43, Jun. 2017, ISSN: 0360-0300. DOI: 10.1145/3057266. [Online]. Available: http://doi.acm.org/10.1145/3057266.

[20] L. Belli, S. Cirani, G. Ferrari, L. Melegari, and M. Picone, "A graph-based cloud architecture for big stream real-time applications in the internet of things," in *Advances in Service-Oriented and Cloud Computing*, G. Ortiz and C. Tran, Eds., Cham: Springer International Publishing, 2015, pp. 91–105.

[21] J. Manyika et al., "Unlocking the potential of the Internet of Things," *McKinsey & Company*, Jun. 2015. [Online]. Available: https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world.

[22] C. V. N. Index, "The zettabyte era–trends and analysis," *Cisco white paper*, June,2017.

[23] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "Iot middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, Feb. 2017, ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2615180.

[24] *Towards a definition of the Internet of Things (IoT) Revision1 - Published 27 MAY 2015*, https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf.

[25] *P2413 wg*, http://grouper.ieee.org/groups/2413/.

[26] *Ieee sa - p2413 - standard for an architectural framework for the internet of things (iot)*, https://standards.ieee.org/develop/project/2413.html.

[27] *Define iot - ieee internet of things*, https://iot.ieee.org/definition.html, (Accessed on 09/09/2017).

[28] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2010.05.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610001568.

[29] CISCO, *Iot reference model white paper june 4, 2014*, http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf, (Accessed on 05/15/2018).

[30] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 586–602, Oct. 2017. DOI: 10.1109/TETC.2016.2606384.

[31] M. Weyrich and C. Ebert, "Reference architectures for the internet of things," *IEEE Software*, vol. 33, no. 1, pp. 112–116, Jan. 2016, ISSN: 0740-7459. DOI: 10.1109/MS.2016.20.

[32] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, Jul. 2015, ISSN: 1573-7705. DOI: 10.1007/s10922-014-9307-7. [Online]. Available: https://doi.org/10.1007/s10922-014-9307-7.

[33] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, May 2010, ISSN: 1869-0238. DOI: 10.1007/s13174-010-0007-6. [Online]. Available: https://doi.org/10.1007/s13174-010-0007-6.

[34] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, Aug. 2016, ISSN: 0018-9162. DOI: 10.1109/MC.2016.245.

[35] Q. Duan, Y. Yan, and A. V. Vasilakos, "A survey on service-oriented network virtualization toward convergence of networking and cloud computing," *IEEE Transactions on Network and Service Management*, vol. 9, no. 4, pp. 373–392, Dec. 2012, ISSN: 1932-4537. DOI: 10.1109/TNSM.2012.113012.120310.

[36] A. V. Dastjerdi and R. Buyya, "Compatibility-aware cloud service composition under fuzzy preferences of users," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 1–13, Jan. 2014, ISSN: 2168-7161. DOI: 10.1109/TCC.2014.2300855.

[37] B. Tang, Z. Chen, G. Hefferman, S. Pei, T. Wei, H. He, and Q. Yang, "Incorporating intelligence in fog computing for big data analysis in smart cities," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 5, pp. 2140–2150, Oct. 2017, ISSN: 1551-3203. DOI: 10.1109/TII.2017.2679740.

[38] *Aws greengrass - embedded lambda compute in connected devices - amazon web services*, https://aws.amazon.com/greengrass/, ()

[39] Microsoft, *Azure iot edge*, https://github.com/Azure/iot-edge-v1, (Accessed on 01/05/2019).

[40] 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*, ser. MCC '12, Helsinki, Finland: ACM, 2012, pp. 13–16, ISBN: 978-1-4503-1519-7. DOI: 10.1145/2342509.2342513. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513.

[41] *OpenFog Reference Architecture : OpenFog Consortium*, https://www.openfogconsortium.org/ra/, (Accessed on 09/13/2017).

[42] R. Mahmud and R. Buyya, "Fog computing: A taxonomy, survey and future directions," *CoRR*, vol. abs/1611.05539, 2016. [Online]. Available: http://arxiv.org/abs/1611.05539.

[43] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, Dec. 2016, ISSN: 0018-9340. DOI: 10.1109/TC.2016.2536019.

[44] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu, and F. Bonomi, "Improving web sites performance using edge servers in fog computing architecture," in *Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, ser. SOSE '13, Washington, DC, USA: IEEE Computer Society, 2013, pp. 320–323, ISBN: 978-0-7695-4944-6. DOI: 10.1109/SOSE.2013.73. [Online]. Available: http://dx.doi.org/10.1109/SOSE.2013.73.

[45] M. S. H. Nazmudeen, A. T. Wan, and S. M. Buhari, "Improved throughput for power line communication (plc) for smart meters using fog computing based data aggregation approach," in *2016 IEEE International Smart Cities Conference (ISC2)*, Sep. 2016, pp. 1–4. DOI: 10.1109/ISC2.2016.7580841.

[46] M. Aazam and E. N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *2014 International Conference on Future Internet of Things and Cloud*, Aug. 2014, pp. 464–470. DOI: 10.1109/FiCloud.2014.83.

[47] S. Cirani, G. Ferrari, N. Iotti, and M. Picone, "The iot hub: A fog node for seamless management of heterogeneous connected smart objects," in *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops)*, Jun. 2015, pp. 1–6. DOI: 10.1109/SECONW.2015.7328145.

[48] C. Dsouza, G. J. Ahn, and M. Taguinod, "Policy-driven security management for fog computing: Preliminary framework and a case study," in *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*, Aug. 2014, pp. 16–23. DOI: 10.1109/IRI.2014.7051866.

[49] V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli, "On qos-aware scheduling of data stream applications over fog computing infrastructures," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, Jul. 2015, pp. 271–276. DOI: 10.1109/ISCC.2015.7405527.

[50] S. Yan, M. Peng, and W. Wang, "User access mode selection in fog computing based radio access networks," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6. DOI: 10.1109/ICC.2016.7510854.

[51] 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, Jan. 2017, ISSN: 2168-6750. DOI: 10.1109/TETC.2015.2508382.

[52] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016, ISSN: 0018-9545. DOI: 10.1109/TVT.2016.2532863.

[53] D. Ye, M. Wu, S. Tang, and R. Yu, "Scalable fog computing with service offloading in bus networks," in *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*, Jun. 2016, pp. 247–251. DOI: 10.1109/CSCloud.2016.34.

[54] IEEE, *Ieee standards association - iot ecosystem study*, http://standards.ieee.org/innovate/iot/study.html, (Accessed on 11/19/2017).

[55] *OpenFog Consortium*, https://www.openfogconsortium.org/, (Accessed on 09/13/2017).

[56] *Industrial internet consortium*, https://www.iiconsortium.org/, (Accessed on 04/05/2020).

[57] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec. 2016, ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2584538.

[58] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta, "The internet of things has a gateway problem," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '15, Santa Fe, New Mexico, USA: ACM, 2015, pp. 27–32, ISBN: 978-1-4503-3391-7. DOI: 10.1145/2699343.2699344. [Online]. Available: http://doi.acm.org/10.1145/2699343.2699344.

[59] *Foghorn systems: Edge intelligence software for iiot*, https://www.foghorn.io/, (Accessed on 11/03/2017).

[60] *Ibm watson - ibm and cisco cloud analytics*, https://www.ibm.com/internet-of-things/partners/ibm-cisco/, (Accessed on 11/03/2017).

[61] M. J. McGrath and C. N. Scanaill, "Sensor network topologies and design considerations," in *Sensor Technologies: Healthcare, Wellness, and Environmental Applications*. Berkeley, CA: Apress, 2013, pp. 79–95, ISBN: 978-1-4302-6014-1. DOI: 10.1007/978-1-4302-6014-1_4. [Online]. Available: https://doi.org/10.1007/978-1-4302-6014-1_4.

[62] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "Ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017. DOI: 10.1002/spe.2509. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2509. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2509.

[63] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Computing*, vol. 26, no. 12, pp. 1519–1534, 2000, Graph Partitioning and Parallel Computing, ISSN: 0167-8191. DOI: https://doi.org/10.1016/S0167-8191(00)00048-X. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016781910000048X.

[64] T. Verbelen, T. Stevens, P. Simoens, F. D. Turck, and B. Dhoedt, "Dynamic deployment and quality adaptation for mobile augmented reality applications," *Journal of Systems and Software*, vol. 84, no. 11, pp. 1871–1882, 2011, Mobile Applications: Status and Trends, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2011.06.063. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016412121100166X.

[65] S. Ou, K. Yang, and J. Zhang, "An effective offloading middleware for pervasive services on mobile devices," *Pervasive and Mobile Computing*, vol. 3, no. 4, pp. 362–385, 2007, Middleware for Pervasive Computing, ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2007.04.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119207000338.

[66] M. Darø Kristensen, "Scavenger: Transparent development of efficient cyber foraging applications," in *2010 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Mar. 2010, pp. 217–226. DOI: 10.1109/PERCOM.2010.5466972.

[67] L. Yang, J. Cao, S. Tang, T. Li, and A. T. S. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *2012 IEEE Fifth International Conference on Cloud Computing*, Jun. 2012, pp. 794–802. DOI: 10.1109/CLOUD.2012.97.

[68] H. Wu, Q. Wang, and K. Wolter, "Mobile healthcare systems with multi-cloud offloading," in *2013 IEEE 14th International Conference on Mobile Data Management*, vol. 2, Jun. 2013, pp. 188–193. DOI: 10.1109/MDM.2013.92.

[69] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, ser. MCS '12, Low Wood Bay, Lake District, UK: Association for Computing Machinery, 2012, pp. 29–36, ISBN: 9781450313193. DOI: 10.1145/2307849.2307858. [Online]. Available: https://doi.org/10.1145/2307849.2307858.

[70] S. Han, S. Zhang, J. Cao, Y. Wen, and Y. Zhang, "A resource aware software partitioning algorithm based on mobility constraints in pervasive grid environments," *Future Generation Computer Systems*, vol. 24, no. 6, pp. 512–529, 2008, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2007.07.013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X07001306.

[71] G. Reiter, "Wireless connectivity for the internet of things-technical report," *Texas Instruments Incorporated, Texas*, [Online]. Available: http://www.ti.com.cn/cn/lit/wp/swry010/swry010.pdf.

[72] G. P. Picco, "Software engineering and wireless sensor networks: Happy marriage or consensual divorce?" In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10, Santa Fe, New Mexico, USA: Association for Computing Machinery, 2010, pp. 283–286, ISBN: 9781450304276. DOI: 10.1145/1882362.1882421. [Online]. Available: https://doi.org/10.1145/1882362.1882421.

[73] B. Morin, N. Harrand, and F. Fleurey, "Model-based software engineering to tame the iot jungle," *IEEE Software*, vol. 34, no. 1, pp. 30–36, Jan. 2017, ISSN: 1937-4194. DOI: 10.1109/MS.2017.11.

[74] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, ser. MCC '13, Hong Kong, China: ACM, 2013, pp. 15–20, ISBN: 978-1-4503-2180-8. DOI: 10.1145/2491266.2491270. [Online]. Available: http://doi.acm.org/10.1145/2491266.2491270.

[75] S. Fowler, *Production-ready microservices : building standardized systems across an engineering organization*. Sebastopol, CA: O'Reilly Media, 2017, ISBN: 978-1491965979.

[76] J. Venkatesh, B. Aksanli, C. S. Chan, A. S. Akyurek, and T. S. Rosing, "Modular and personalized smart health application design in a smart city environment," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 614–623, 2018.

[77] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2016, pp. 1–6. DOI: 10.1109/ETFA.2016.7733707.

[78] *SmartEdge: fog computing cloud extensions to support latency-sensitive IoT applications*, https://repositorio.ufrn.br/jspui/handle/123456789/22557, (Accessed on 09/13/2017), Dec. 2016.

[79] M. Yannuzzi, F. van Lingen, A. Jain, O. L. Parellada, M. M. Flores, D. Carrera, J. L. Pérez, D. Montero, P. Chacin, A. Corsaro, and A. Olive, "A new era for cities with fog computing," *IEEE Internet Computing*, vol. 21, no. 2, pp. 54–67, Mar. 2017, ISSN: 1089-7801. DOI: 10.1109/MIC.2017.25.

[80] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a smart city internet of things platform with microservice architecture," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Aug. 2015, pp. 25–30. DOI: 10.1109/FiCloud.2015.55.

[81] I. S. Udoh and G. Kotonya, "Developing iot applications: Challenges and frameworks," *IET Cyber-Physical Systems: Theory Applications*, vol. 3, no. 2, pp. 65–72, 2018, ISSN: 2398-3396. DOI: 10.1049/iet-cps.2017.0068.

[82] S. Nastic, S. Sehic, M. Vögler, H. L. Truong, and S. Dustdar, "Patricia – a novel programming model for iot applications on cloud platforms," in *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, Dec. 2013, pp. 53–60. DOI: 10.1109/SOCA.2013.48.

[83] B. Costa, P. F. Pires, and F. C. Delicato, "Modeling iot applications with sysml4iot," in *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug. 2016, pp. 157–164. DOI: 10.1109/SEAA.2016.19.

[84] X. T. Nguyen, H. T. Tran, H. Baraki, and K. Geihs, "Frasad: A framework for model-driven iot application development," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec. 2015, pp. 387–392. DOI: 10.1109/WF-IoT.2015.7389085.

[85] S. K. Datta and C. Bonnet, "Easing iot application development through datatweet framework," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec. 2016, pp. 430–435. DOI: 10.1109/WF-IoT.2016.7845390.

[86] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '16, Irvine, California: Association for Computing Machinery, 2016, pp. 258–269, ISBN: 9781450340212. DOI: 10.1145/2933267.2933317. [Online]. Available: https://doi.org/10.1145/2933267.2933317.

[87] N. K. Giang, M. Blackstock, R. Lea, and V. C. M. Leung, "Developing iot applications in the fog: A distributed dataflow approach," in *2015 5th International Conference on the Internet of Things (IOT)*, Oct. 2015, pp. 155–162. DOI: 10.1109/IOT.2015.7356560.

[88] N. K. Giang, M. Blackstock, R. Lea, and V. C. M. Leung, "Distributed data flow: A programming model for the crowdsourced internet of things," in *Proceedings of the Doctoral Symposium of the 16th International Middleware Conference*, ser. Middleware Doct Symposium '15, Vancouver, BC, Canada: ACM, 2015, 4:1–4:4, ISBN: 978-1-4503-3728-1. DOI: 10.1145/2843966.2843970. [Online]. Available: http://doi.acm.org/10.1145/2843966.2843970.

[89] W. M. Johnston, J. R. P. Hanna, and R. J. Millar, "Advances in dataflow programming languages," *ACM Comput. Surv.*, vol. 36, no. 1, pp. 1–34, Mar. 2004, ISSN: 0360-0300. DOI: 10.1145/1013208.1013209. [Online]. Available: https://doi.org/10.1145/1013208.1013209.

[90] G. Mainland, G. Morrisett, M. Welsh, and R. Newton, "Sensor network programming with flask," in *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '07, Sydney, Australia: Association for Computing Machinery, 2007, pp. 385–386, ISBN: 9781595937636. DOI: 10.1145/1322263.1322307. [Online]. Available: https://doi.org/10.1145/1322263.1322307.

[91] K. McLaughlin, *Gaps in 4g network hinder high-tech agriculture; fcc prepares to release $500 million to improve coverage*, http://www.bendbulletin.com/home/4535283-151/gaps-in-4g-network-hinder-high-tech-agriculture, (Accessed on 12/02/2017).

[92] *Aggateway*, http://www.aggateway.org/, (Accessed on 12/03/2017).

[93] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman, "Farmbeats: An iot platform for data-driven agriculture," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA: USENIX Association, 2017, pp. 515–529, ISBN: 978-1-931971-37-9. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vasisht.

[94] A. (Pentland, R. Fletcher, and A. Hasson, "Daknet: Rethinking connectivity in developing nations," *Computer*, vol. 37, no. 1, pp. 78–83, Jan. 2004, ISSN: 0018-9162. DOI: 10.1109/MC.2004.1260729. [Online]. Available: http://dx.doi.org/10.1109/MC.2004.1260729.

[95] Q. Ag, *Cattle health management, tags, and software | quantified ag*, http://quantifiedag.com/, (Accessed on 12/07/2017).

[96] D. Ireland, *Milking parlours & machines, heat detection, scrapers, feeders, milk cooling - dairymaster ireland*, http://www.dairymaster.ie/, (Accessed on 12/07/2017).

[97] *Boumatic*, https://boumatic.com/us_en/, (Accessed on 12/07/2017).

[98] K. Taylor, C. Griffith, L. Lefort, R. Gaire, M. Compton, T. Wark, D. Lamb, G. Falzon, and M. Trotter, "Farming the web of things," *IEEE Intelligent Systems*, vol. 28, no. 6, pp. 12–19, Nov. 2013, ISSN: 1541-1672. DOI: 10.1109/MIS.2013.102.

[99] M.-C. Chen, C.-H. Chen, and C.-Y. Siang, "Design of information system for milking dairy cattle and detection of mastitis," *Mathematical Problems in Engineering*, vol. 2014, pp. 1–9, 2014. DOI: 10.1155/2014/759019.

[100] T. Wark, D. Swain, C. Crossman, P. Valencia, G. Bishop-Hurley, and R. Handcock, "Sensor and actuator networks: Protecting environmentally sensitive areas," *IEEE Pervasive Computing*, vol. 8, no. 1, pp. 30–36, Jan. 2009, ISSN: 1536-1268. DOI: 10.1109/MPRV.2009.15.

[101] C. Brewster, I. Roussaki, N. Kalatzis, K. Doolin, and K. Ellis, "Iot in agriculture: Designing a europe-wide large-scale pilot," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 26–33, 2017, ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1600528.

[102] W. Steeneveld and H. Hogeveen, "Characterization of dutch dairy farms using sensor systems for cow management," *Journal of Dairy Science*, vol. 98, no. 1, pp. 709–717, 2015, ISSN: 0022-0302. DOI: http://dx.doi.org/10.3168/jds.2014-8595. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022030214007863.

[103] I. Andonovic, C. Michie, M. Gilroy, H. G. Goh, K. H. Kwong, K. Sasloglou, and T. Wu, "Wireless sensor networks for cattle health monitoring," in *ICT Innovations 2009*, D. Davcev and J. M. Gómez, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 21–31, ISBN: 978-3-642-10781-8. DOI: 10.1007/978-3-642-10781-8_3. [Online]. Available: https://doi.org/10.1007/978-3-642-10781-8_3.

[104] P. Schmitt, D. Iland, M. Zheleva, and E. Belding, "Hybridcell: Cellular connectivity on the fringes with demand-driven local cells," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, Apr. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524591.

[105] K. Heimerl, K. Ali, J. Blumenstock, B. Gawalt, and E. Brewer, "Expanding rural cellular networks with virtual coverage," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, Lombard, IL: USENIX Association, 2013, pp. 283–296, ISBN: 978-1-931971-00-3. [Online]. Available: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/heimurl.

[106] J. A. Vázquez Diosdado, Z. E. Barker, H. R. Hodges, J. R. Amory, D. P. Croft, N. J. Bell, and E. A. Codling, "Classification of behaviour in housed dairy cows using an accelerometer-based activity monitoring system," *Animal Biotelemetry*, vol. 3, no. 1, p. 15, Jun. 2015, ISSN: 2050-3385. DOI: 10.1186/s40317-015-0045-8. [Online]. Available: https://doi.org/10.1186/s40317-015-0045-8.

[107] N. Gandhi and L. J. Armstrong, "A review of the application of data mining techniques for decision making in agriculture," in *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, Dec. 2016, pp. 1–6. DOI: 10.1109/IC3I.2016.7917925.

[108] M. Williams, N. M. Parthaláin, P. Brewer, W. James, and M. Rose, "A novel behavioral model of the pasture-based dairy cow from gps data using data mining and machine learning techniques," *Journal of Dairy Science*, vol. 99, no. 3, pp. 2063–2075, 2016, ISSN: 0022-0302. DOI: https://doi.org/10.3168/jds.2015-10254. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022030216000667.

[109] D. S. Zois and U. Mitra, "Controlled sensing: A myopic fisher information sensor selection algorithm," in *2014 IEEE Global Communications Conference*, Dec. 2014, pp. 3401–3406. DOI: 10.1109/GLOCOM.2014.7037333.

[110] D. S. Zois, M. Levorato, and U. Mitra, "Heterogeneous time-resource allocation in wireless body area networks for green, maximum likelihood activity detection," in *2012 IEEE International Conference on Communications (ICC)*, Jun. 2012, pp. 3399–3403. DOI: 10.1109/ICC.2012.6364460.

[111] D. S. Zois, M. Levorato, and U. Mitra, "Active classification for pomdps: A kalman-like state estimator," *IEEE Transactions on Signal Processing*, vol. 62, no. 23, pp. 6209–6224, Dec. 2014, ISSN: 1053-587X. DOI: 10.1109/TSP.2014.2362098.

[112] S. Wolfert, L. Ge, C. Verdouw, and M.-J. Bogaardt, "Big data in smart farming – a review," *Agricultural Systems*, vol. 153, pp. 69–80, 2017, ISSN: 0308-521X. DOI: http://dx.doi.org/10.1016/j.agsy.2017.01.023. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0308521X16303754.

[113] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, Fourthquarter 2015, ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2444095.

[114] C. Rutten, A. Velthuis, W. Steeneveld, and H. Hogeveen, "Invited review: Sensors to support health management on dairy farms," *Journal of Dairy Science*, vol. 96, no. 4, pp. 1928–1952, 2013, ISSN: 0022-0302. DOI: http://dx.doi.org/10.3168/jds.2012-6107. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022030213001409.

[115] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016, ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2579198.

[116] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in *2014 Federated Conference on Computer Science and Information Systems*, Sep. 2014, pp. 1–8. DOI: 10.15439/2014F503.

[117] *NASA Sees Hurricane Ophelia Lashing Ireland | NASA*, https://www.nasa.gov/feature/goddard/2017/ophelia-atlantic-ocean, (Accessed on 12/03/2017).

[118] *Hurricane ophelia hits ireland - live tracker shows what the weather will be like where you live - irish mirror online*, http://www.irishmirror.ie/news/irish-news/hurricane-ophelia-hits-ireland-live-11349140, (Accessed on 12/03/2017).

[119] M. Caria, J. Schudrowitz, A. Jukan, and N. Kemper, "Smart farm computing systems for animal welfare monitoring," in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2017, pp. 152–157. DOI: 10.23919/MIPRO.2017.7973408.

[120] *Raspberry Pi - Teach, Learn, and Make with Raspberry Pi*, https://www.raspberrypi.org/, (Accessed on 09/06/2017).

[121] T.-C. Hsu, H. Yang, Y.-C. Chung, and C.-H. Hsu, "A creative iot agriculture platform for cloud fog computing," *Sustainable Computing: Informatics and Systems*, 2018, ISSN: 2210-5379. DOI: https://doi.org/10.1016/j.suscom.2018.10.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2210537918303275.

[122] M. A. Zamora-Izquierdo, J. Santa, J. A. Martínez, V. Martínez, and A. F. Skarmeta, "Smart farming iot platform based on edge and cloud computing," *Biosystems Engineering*, vol. 177, pp. 4–17, 2019, Intelligent Systems for Environmental Applications, ISSN: 1537-5110. DOI: https://doi.org/10.1016/j.biosystemseng.2018.10.014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1537511018301211.

[123] J. Muangprathub, N. Boonnam, S. Kajornkasirat, N. Lekbangpong, A. Wanichsombat, and P. Nillaor, "Iot and agriculture data analysis for smart farm," *Computers and Electronics in Agriculture*, vol. 156, pp. 467–474, 2019, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2018.12.011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169918308913.

[124] A. Jukan, X. Masip-Bruin, and N. Amla, "Smart computing and sensing technologies for animal welfare: A systematic review," *ACM Comput. Surv.*, vol. 50, no. 1, 10:1–10:27, Apr. 2017, ISSN: 0360-0300. DOI: 10.1145/3041960. [Online]. Available: http://doi.acm.org/10.1145/3041960.

[125] M. Zheleva, P. Bogdanov, D.-S. Zois, W. Xiong, R. Chandra, and M. Kimball, "Smallholder agriculture in the information age: Limits and opportunities," in *Proceedings of the 2017 Workshop on Computing Within Limits*, ser. LIMITS '17, Santa Barbara, California, USA: ACM, 2017, pp. 59–70, ISBN: 978-1-4503-4950-5. DOI: 10.1145/3080556.3080563. [Online]. Available: http://doi.acm.org/10.1145/3080556.3080563.

[126] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '14, Bretton Woods, New Hampshire, USA: Association for Computing Machinery, 2014, pp. 68–81, ISBN: 9781450327930. DOI: 10.1145/2594368.2594383. [Online]. Available: https://doi.org/10.1145/2594368.2594383.

[127] F. AIT SALAHT, F. Desprez, and A. Lebre, "An overview of service placement problem in Fog and Edge Computing," Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, LYON, France, Research Report RR-9295, Oct. 2019, pp. 1–43. [Online]. Available: https://hal.inria.fr/hal-02313711.

[128] H. R. Arkian, A. Diyanat, and A. Pourkhalili, "Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications," *Journal of Network and Computer Applications*, vol. 82, pp. 152–165, 2017, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.01.012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517300188.

[129] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, Mar. 2017, ISSN: 2372-2568. DOI: 10.1109/MCC.2017.27.

[130] J. K. Zao, T. T. Gan, C. K. You, S. J. R. Méndez, C. E. Chung, Y. T. Wang, T. Mullen, and T. P. Jung, "Augmented Brain Computer Interaction Based on Fog Computing and Linked Data," in *Intelligent Environments (IE), 2014 International Conference on*, Jun. 2014, pp. 374–377. DOI: 10.1109/IE.2014.54.

[131] T. N. Gia, M. Jiang, A. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare internet of things: A case study on ecg feature extraction," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, Oct. 2015, pp. 356–363, ISBN: 9781509001545. DOI: 10.1109/CIT/IUCC/DASC/PICOM.2015.51.

[132] I. Sommerville, *Software Engineering*, 10th. Pearson, 2015, ISBN: 0133943038.

[133] Y. Jarraya, A. Eghtesadi, M. Debbabi, Y. Zhang, and M. Pourzandi, "Cloud calculus: Security verification in elastic cloud computing platform," in *2012 International Conference on Collaboration Technologies and Systems (CTS)*, May 2012, pp. 447–454. DOI: 10.1109/CTS.2012.6261089.

[134] E. Kavvadia, S. Sagiadinos, K. Oikonomou, G. Tsioutsiouliklis, and S. Aïssa, "Elastic virtual machine placement in cloud computing network environments," *Computer Networks*, vol. 93, pp. 435–447, 2015, Cloud Networking and Communications II, ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.comnet.2015.09.038. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615003631.

[135] K. Velasquez, D. P. Abreu, M. Curado, and E. Monteiro, "Service placement for latency reduction in the internet of things," English, *Annals of Telecommunications*, vol. 72, no. 1-2, pp. 105–115, Jun. 2017. DOI: 10.1007/s12243-016-0524-9.

[136] A. Brogi, A. Ibrahim, J. Soldani, J. Carrasco, J. Cubo, E. Pimentel, and F. D'Andria, "Seaclouds: A european project on seamless management of multi-cloud applications," *SIGSOFT Softw. Eng. Notes*, vol. 39, no. 1, pp. 1–4, Feb. 2014, ISSN: 0163-5948. DOI: 10.1145/2557833.2557844. [Online]. Available: https://doi.org/10.1145/2557833.2557844.

[137] A. Corradi, L. Foschini, A. Pernafini, F. Bosi, V. Laudizio, and M. Seralessandri, "Cloud paas brokering in action: The cloud4soa management infrastructure," in *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*, Sep. 2015, pp. 1–7. DOI: 10.1109/VTCFall.2015.7390868.

[138] *Powerful infrastructure automation and delivery | puppet | puppet.com*, https://puppet.com/, (Accessed on 02/09/2020).

[139] *Chef infra | chef*, https://www.chef.io/products/chef-infra/, (Accessed on 02/09/2020).

[140] X. Huang, S. Ganapathy, and T. Wolf, "Evaluating algorithms for composable service placement in computer networks," in *2009 IEEE International Conference on Communications*, Jun. 2009, pp. 1–6. DOI: 10.1109/ICC.2009.5199007.

[141] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *2009 International Conference on High Performance Computing Simulation*, Jun. 2009, pp. 1–11. DOI: 10.1109/HPCSIM.2009.5192685.

[142] H. Hong, P. Tsai, and C. Hsu, "Dynamic module deployment in a fog computing platform," in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Oct. 2016, pp. 1–6. DOI: 10.1109/APNOMS.2016.7737202.

[143] V. B. C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, and G. Tashakor, "Handling service allocation in combined fog-cloud scenarios," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–5. DOI: 10.1109/ICC.2016.7511465.

[144] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applications," in *Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '16, Irvine, California: Association for Computing Machinery, 2016, pp. 69–80, ISBN: 9781450340212. DOI: 10.1145/2933267.2933312. [Online]. Available: https://doi.org/10.1145/2933267.2933312.

[145] S. Misra, "Theoretical modelling of fog computing: A green computing paradigm to support iot applications," English, *IET Networks*, vol. 5, 23–29(6), 2 Mar. 2016, ISSN: 2047-4954. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/iet-net.2015.0034.

[146] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds., Curran Associates, Inc., 2008, pp. 161–168. [Online]. Available: http://papers.nips.cc/paper/3323-the-tradeoffs-of-large-scale-learning.pdf.

[147] E. Siow, T. Tiropanis, and W. Hall, "Analytics for the internet of things: A survey," *ACM Comput. Surv.*, vol. 51, no. 4, 74:1–74:36, Jul. 2018, ISSN: 0360-0300. DOI: 10.1145/3204947. [Online]. Available: http://doi.acm.org/10.1145/3204947.

[148] H. Jiang, S. Jin, and C. Wang, "Prediction or not? an energy-efficient framework for clustering-based data collection in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 1064–1071, Jun. 2011, ISSN: 1045-9219. DOI: 10.1109/TPDS.2010.174.

[149] R. V. Kulkarni, A. Forster, and G. K. Venayagamoorthy, "Computational intelligence in wireless sensor networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 13, no. 1, pp. 68–96, First 2011, ISSN: 1553-877X. DOI: 10.1109/SURV.2011.040310.00002.

[150] N. Cheng, N. Lu, N. Zhang, T. Yang, X. (Shen, and J. W. Mark, "Vehicle-assisted device-to-device data delivery for smart grid," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 4, pp. 2325–2340, Apr. 2016, ISSN: 0018-9545. DOI: 10.1109/TVT.2015.2415512.

[151] C. Anagnostopoulos, "Time-optimized contextual information forwarding in mobile sensor networks," *Journal of Parallel and Distributed Computing*, vol. 74, no. 5, pp. 2317–2332, 2014, ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2014.01.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514000185.

[152] O. B. Sezer, E. Dogdu, and A. M. Ozbayoglu, "Context-aware computing, learning, and big data in internet of things: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 1–27, Feb. 2018, ISSN: 2327-4662. DOI: 10.1109/JIOT.2017.2773600.

[153] L. Shi, L. Zhao, W. Song, G. Kamath, Y. Wu, and X. Liu, "Distributed least-squares iterative methods in networks: A survey," *CoRR*, vol. abs/1706.07098, 2017. arXiv: 1706.07098. [Online]. Available: http://arxiv.org/abs/1706.07098.

[154] A. Lazerson, D. Keren, and A. Schuster, "Lightweight monitoring of distributed streams," *ACM Trans. Database Syst.*, vol. 43, no. 2, 9:1–9:37, Jul. 2018, ISSN: 0362-5915. DOI: 10.1145/3226113. [Online]. Available: http://doi.acm.org/10.1145/3226113.

[155] H. Wang and C. Li, "Distributed quantile regression over sensor networks," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 2, pp. 338–348, Jun. 2018, ISSN: 2373-776X. DOI: 10.1109/TSIPN.2017.2699923.

[156] M. Gabel, D. Keren, and A. Schuster, "Monitoring least squares models of distributed streams," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15, Sydney, NSW, Australia: ACM, 2015, pp. 319–328, ISBN: 978-1-4503-3664-2. DOI: 10.1145/2783258.2783349. [Online]. Available: http://doi.acm.org/10.1145/2783258.2783349.

[157] G. Kamath, P. Agnihotri, M. Valero, K. Sarker, and W. Song, "Pushing analytics to the edge," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2016, pp. 1–6. DOI: 10.1109/GLOCOM.2016.7842181.

[158] *Beagleboard.org - community supported open hardware computers for making*, https://beagleboard.org/, (Accessed on 04/05/2020).

[159] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco, "Practical data prediction for real-world wireless sensor networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2231–2244, Aug. 2015, ISSN: 1041-4347. DOI: 10.1109/TKDE.2015.2411594.

[160] N. Harth and C. Anagnostopoulos, "Quality-aware aggregation & predictive analytics at the edge," in *2017 IEEE International Conference on Big Data (Big Data)*, Dec. 2017, pp. 17–26. DOI: 10.1109/BigData.2017.8257907.

[161] T.-C. Chang, L. Zheng, M. Gorlatova, C. Gitau, C.-Y. Huang, and M. Chiang, "Decomposing data analytics in fog networks," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '17, Delft, Netherlands: ACM, 2017, 35:1–35:2, ISBN: 978-1-4503-5459-2. DOI: 10.1145/3131672.3136962. [Online]. Available: http://doi.acm.org/10.1145/3131672.3136962.

[162] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, ser. NIPS'06, Canada: MIT Press, 2006, pp. 281–288. [Online]. Available: http://dl.acm.org/citation.cfm?id=2976456.2976492.

[163] A. Van Nuffel, I. Zwertvaegher, L. Pluym, S. Van Weyenberg, V. M. Thorup, M. Pastell, B. Sonck, and W. Saeys, "Lameness detection in dairy cows: Part 1. how to distinguish between non-lame and lame cows based on differences in locomotion or behavior," *Animals*, vol. 5, no. 3, pp. 838–860, 2015, ISSN: 2076-2615. DOI: 10.3390/ani5030387. [Online]. Available: http://www.mdpi.com/2076-2615/5/3/387.

[164] *Economic cost of lameness in irish dairy herds*, https://www.xlvets.ie/sites/xlvets.ie/files/press-article-files/XLVets%2520Article%2520Forage%2520Guide%25202012.pdf, (Accessed on 05/07/2019).

[165] *MQTT*, http://mqtt.org/, (Last Accessed on August 03,2017).

[166] *Home | amqp*, https://www.amqp.org/, (Accessed on 04/06/2020).

[167] *Xmpp | xmpp main*, https://xmpp.org/, (Accessed on 04/06/2020).

[168] *Getting to know MQTT*, https://www.ibm.com/developerworks/library/iot-mqtt-why-good-for-iot/index.html, (Last accessed on August 03,2017).

[169] S. Lee, H. Kim, D. k. Hong, and H. Ju, "Correlation analysis of mqtt loss and delay according to qos level," in *The International Conference on Information Networking 2013 (ICOIN)*, Jan. 2013, pp. 714–717. DOI: 10.1109/ICOIN.2013.6496715.

[170] *Mqtt essentials part 6: Quality of service 0, 1 & 2*, https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/, (Accessed on 03/12/2019).

[171] *Empowering app development for developers | Docker*, https://www.docker.com/.

[172] *Ionic framework - ionic documentation*, https://ionicframework.com/docs, (Accessed on 02/24/2020).

[173] *PouchDB, the JavaScript Database that Syncs!* https://pouchdb.com/, (Accessed on 05/02/2019).

[174] M. Taneja, N.Jalodia, J. Byabazaire, A. Davy, and C. Olariu, *SmartHerd-Connected Cows Demo - Google Drive*, (Accessed on 10/04/2018). [Online]. Available: https://drive.google.com/file/d/1QIrKSp8SkAZRRAFQDQHdPXu1TVYaVyv3/view.

[175] *psutil – PyPI*, https://pypi.org/project/psutil/, (Accessed on 03/13/2019).

[176] Intel, *Intel® Power Gadget | Intel®Software*, https://software.intel.com/en-us/articles/intel-power-gadget-20, (Accessed on 03/14/2019).

[177] *Cgroups(7) - linux manual page*, http://man7.org/linux/man-pages/man7/cgroups.7.html, (Accessed on 03/14/2020).

[178] *Home - chaos monkey*, https://netflix.github.io/chaosmonkey/, (Accessed on 03/13/2020).

[179] *Response time limits: Article by jakob nielsen*, https://www.nngroup.com/articles/response-times-3-important-limits/, (Accessed on 03/05/2020).

[180] B. Davies, *Exploring chaos: Theory and experiment*. CRC Press, 2018.

[181] A. Adewumi, J. Kagamba, and A. Alochukwu, "Application of chaos theory in the prediction of motorised traffic flows on urban networks," *Mathematical Problems in Engineering*, vol. 2016, 2016.

[182] *What is scalability? - definition from techopedia*, https://www.techopedia.com/definition/9269/scalability, (Accessed on 03/04/2020).

[183] J. Venkatesh, B. Aksanli, C. S. Chan, A. S. Akyürek, and T. S. Rosing, "Scalable-application design for the iot," *IEEE Software*, vol. 34, no. 1, pp. 62–70, Jan. 2017, ISSN: 1937-4194. DOI: 10.1109/MS.2017.4.

[184] M. B. Stephenson and D. W. Bailey, "Do movement patterns of gps-tracked cattle on extensive rangelands suggest independence among individuals?" *Agriculture*, vol. 7, no. 7, 2017, ISSN: 2077-0472. DOI: 10.3390/agriculture7070058. [Online]. Available: http://www.mdpi.com/2077-0472/7/7/58.

[185] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[186] A. Brogi, S. Forti, C. Guerrero, and I. Lera, "How to place your apps in the fog: State of the art and open challenges," *Software: Practice and Experience*, vol. n/a, no. n/a, DOI: 10.1002/spe.2766. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2766. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2766.

[187] A. Brogi, S. Forti, and A. Ibrahim, "How to best deploy your fog applications, probably," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, May 2017, pp. 105–114. DOI: 10.1109/ICFEC.2017.8.

[188] N. Helwig, E. Pignanelli, and A. Schütze, "Condition monitoring of a complex hydraulic system using multivariate statistics," in *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*, May 2015, pp. 210–215. DOI: 10.1109/I2MTC.2015.7151267.

[189] M. Kuhn and K. Johnson, *Applied predictive modeling*. Springer-Verlag New York, 2013, vol. 26.

[190] A. R. Benson, D. F. Gleich, and J. Demmel, "Direct QR factorizations for tall-and-skinny matrices in mapreduce architectures," *CoRR*, vol. abs/1301.1071, 2013. arXiv: 1301.1071. [Online]. Available: http://arxiv.org/abs/1301.1071.

[191] C. Misra, S. Haldar, S. Bhattacharya, and S. K. Ghosh, "Spin: A fast and scalable matrix inversion method in apache spark," in *Proceedings of the 19th International Conference on Distributed Computing and Networking*, ser. ICDCN '18, Varanasi, India: ACM, 2018, 16:1–16:10, ISBN: 978-1-4503-6372-3. DOI: 10.1145/3154273.3154300. [Online]. Available: http://doi.acm.org/10.1145/3154273.3154300.

[192] M. Benzi, "Preconditioning techniques for large linear systems: A survey," *J. Comput. Phys.*, vol. 182, no. 2, pp. 418–477, Nov. 2002, ISSN: 0021-9991. DOI: 10.1006/jcph.2002.7176. [Online]. Available: http://dx.doi.org/10.1006/jcph.2002.7176.

[193] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003, ISBN: 0898715342.

[194] M. Kearns, "Efficient noise-tolerant learning from statistical queries," *J. ACM*, vol. 45, no. 6, pp. 983–1006, Nov. 1998, ISSN: 0004-5411. DOI: 10.1145/293347.293351. [Online]. Available: http://doi.acm.org/10.1145/293347.293351.

[195] M. Kearns, "Efficient noise-tolerant learning from statistical queries," in *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '93, San Diego, California, USA: ACM, 1993, pp. 392–401, ISBN: 0-89791-591-7. DOI: 10.1145/167088.167200. [Online]. Available: http://doi.acm.org/10.1145/167088.167200.

[196] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, Nov. 1984, ISSN: 0001-0782. DOI: 10.1145/1968.1972. [Online]. Available: http://doi.acm.org/10.1145/1968.1972.

[197] Intel, *Intel® IoT Gateway*, https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/gateway-solutions-iot-brief.pdf, (Accessed on 03/12/2019).

[198] Dell, *Edge gateway 5100 | dell ireland*, https://goo.gl/vWcdSW, (Accessed on 03/12/2019).

[199] *Uci machine learning repository: Air quality data set*, http://archive.ics.uci.edu/ml/datasets/air+quality, (Accessed on 12/24/2018).

[200] S. D. Vito, E. Massera, M. Piga, L. Martinotto, and G. D. Francia, "On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario," *Sensors and Actuators B: Chemical*, vol. 129, no. 2, pp. 750–757, 2008, ISSN: 0925-4005. DOI: https://doi.org/10.1016/j.snb.2007.09.060. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925400507007691.

[201] *Eclipse paho - mqtt*, https://www.eclipse.org/paho/.

[202] *Eclipse mosquitto -mqtt*, https://mosquitto.org/.

[203] *Github - xybu/python-resmon*, https://github.com/xybu/python-resmon.

[204] *Glances - An Eye on your system*, https://nicolargo.github.io/glances/.

[205] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, Nov. 2018, ISSN: 1533-5399. DOI: 10.1145/3186592. [Online]. Available: https://doi.org/10.1145/3186592.

[206] S. Venticinque and A. Amato, "A methodology for deployment of iot application in fog," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 5, pp. 1955–1976, 2019.

[207] 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, Oct. 2017, ISSN: 2372-2541. DOI: 10.1109/JIOT.2017.2701408.

Appendix A

Probabilistic Analysis of System Fault Tolerance and Resilience

Over here, we present the empirical mathematical analysis using the concepts from probability and combination theory for such a behaviour by the developed software solution during the fault tolerance experiment. We start with the analysis on the fault tolerance experiment, and further lead the analysis to System Resilience feature of the developed IoT solution. We also see that how the fog functionality of the developed solution adds to the trait of fault tolerance and resilience. For convenience, the notations used have been listed in Table A.1 for quick reference.

Table A.1 Symbols and Notations Used.

Symbol	Interpretation
n	Total number of microservices that make the system
k	Number of services out of n that are getting broken
2^n	Total possible system states, given that there are two possible states for each service i.e., either working or broken (Set T = Total possible different combinations of broken and working services together, and $\|T\| = 2^n$
p	Probability of a service getting broken
2^k	Total possible combination for broken services to come back into the system, this is called set B, and $\|B\| = 2^k$
2^{n-k}	Total possible system states when k services out of n are not working

Assume that we have a total of n microservices that make up the entire system. Now given that the services can get broken in the system leads to two possible states of an

individual service, which are broken (not-working, unavailable, interrupted) or not-broken (working, available, uninterrupted). With n total services, the total possible states of the system are 2^n.

Out of n services, let's say k services are getting broken at a given time. Let's say p is the probability of a service being down or broken, correspondingly $1-p$ is the probability of a service not being down or broken. Then using the concepts of binomial distribution, the probability mass function for k services being down can be written as below:

$$P(X=k) = {}^{n}C_k \times p^k \times (1-p)^{n-k} \tag{A.1}$$

Where $k = 0, 1, 2....n$ and ${}^{n}C_k = \frac{n!}{k!(n-k)!}$. The random variable X here follows the binomial distribution with parameters $n \in \mathbb{N}$ and $p \in [0,1]$. Equation A.1 represents the probability mass function for the system with k services failures.

Now as a service can either be in broken state or in working state, so $p = 0.5$ here. Although, if there is a biasing in the system or for some reason some services have more chances of being broken than others, then p can take the corresponding values accordingly.

In our scenario, keeping $p = 0.5$ in equation A.1 leads to:

$$P(X=k) = {}^{n}C_k \times (0.5)^k \times (0.5)^{n-k} \tag{A.2}$$

Which on solving further leads to equation A.3 below:

$$P(X=k) = \frac{{}^{n}C_k}{2^n} \tag{A.3}$$

The equation A.3 represents the probability of k services getting broken out of a total of n services where each individual service can have two states(available or unavailable), and the whole system can have 2^n possible system states.

Now, we make a plot presented in Fig. A.1 for probability mass function for the binomial distribution with keeping $p = 0.5$ and varying the value of n. Each point on X-axis represents a value corresponding to k. We infer from the plot that as n increases the probability corresponding to k services getting broken as presented on Y-axis decreases.

Which implies that a system built with more number of services has less chances of failures, or a distributed system will have less chances of failures as compared to a centralized system. This is in-turn the trait of a distributed system design i.e., distributed systems are more fault tolerant in nature. This can also be inferred from equation A.3, by substituting the value of $k = n$, which gives :

$$P(X=k) = \frac{{}^{n}C_n}{2^n} = \frac{1}{2^n} \tag{A.4}$$

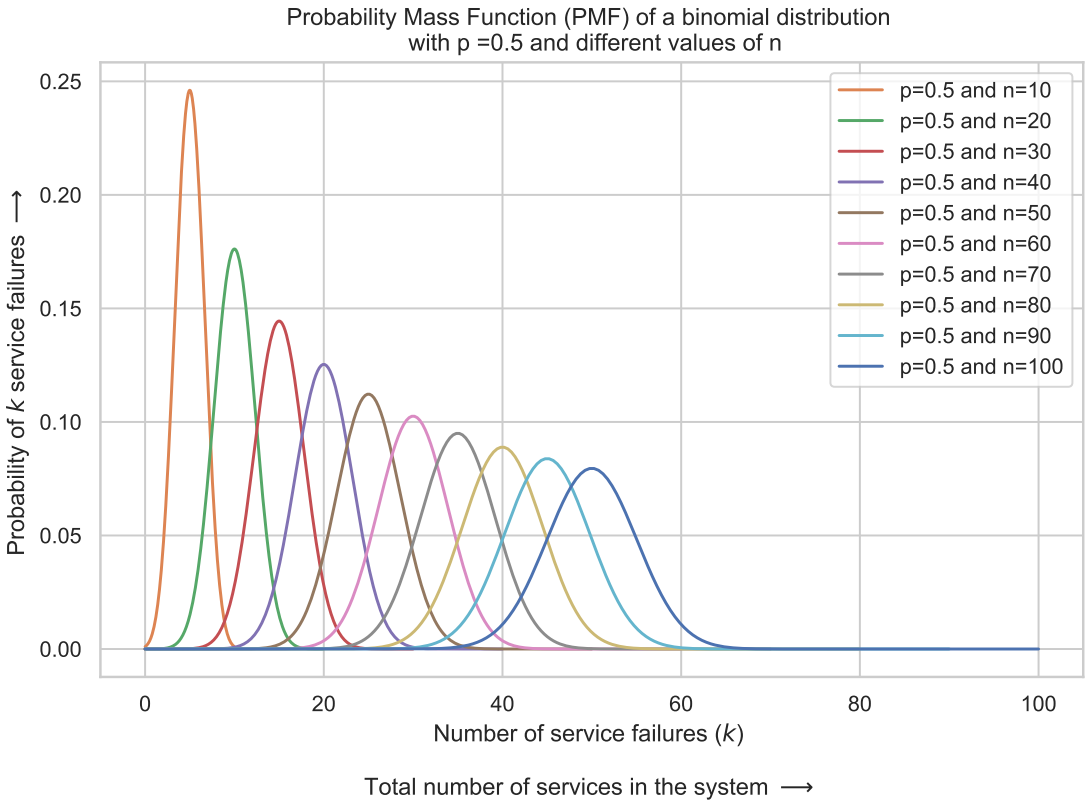


Fig. A.1 Plot presenting probability mass function(pmf) of binomial distribution with p=0.5 and different values of n. We map this pmf with the number of service failures and probability of such service failures in the system.

We can see that there is only one possible system state where none of the services are working (i.e., all the services are broken). But then the probability for such an event to happen is quite less. And as n increases, 2^n also increases and the value becomes even less. Further to note that in production level system, there are multiple instances of a single service that are used in order to achieve efficiency and scalability, which further increases the number of elements in the set of total possible states of the system. Moreover, the services and service instance might also be running in different location i.e., in multiple VMs (Virtual Machines), in geographically multiple locations such as multiple data centres and multiple fog nodes, in multiple VMs in multiple data centres or computational nodes, all this will further increase the number of possible states of the system.

The above postulate supports the approach proposed for designing and developing fog enabled IoT software system proposed in the dissertation, which is to have a hybrid distributed modular application architecture approach using microservices.

System Resilience: Resilience refers to a system's ability to withstand or recover quickly from difficult conditions. It is the capacity to recover quickly from difficulties.

Once again we refer to Table A.1 for the notations used. Let's name the set consisting of total possible system states as T, and another set B which represents the total possible combinations of broken services to come back in the system at a given time. The same has been presented in equation below:

$$T = \text{Set of total possible system states} = 2^n \tag{A.5}$$

$$B = \text{Set of total possible combinations for broken services to come back into the system} = 2^k \tag{A.6}$$

The set B refers to how the broken services come back in working state in the system. To further understand the context behind defining the set B, let's assume 2 services out of n were broken. Now there is a possibility that either both the broken services come back in working state into the system at the same time, or they come one after the other, or none of them come back in working state at all. For a better representation here, the Table A.2 presents the possible combinations, where 0 refers to broken state, and 1 corresponds to the working state. On careful examination, we find that B is a subset of T i.e.,

$$B \subset T \tag{A.7}$$

Table A.2 Possible ways for two broken services to come into working state to join back in the pool of working services.

0	0
0	1
1	0
1	1

Fig. A.2 gives a pictorial representation of the subset relation written in equation A.7. We can see that there will be an exact mapping for a point in set B to a point in set T.

For the system resilience trait there are two possibilities as per the discussion above:

1. The broken services come back into working state into the system one after the other. A diagrammatic representation of the same has been given in Fig. A.3. Over here, the relationship listed in equation A.7 will still hold true for the possible states for the $n-k+1$ working services and $k-1$ services that are not working.

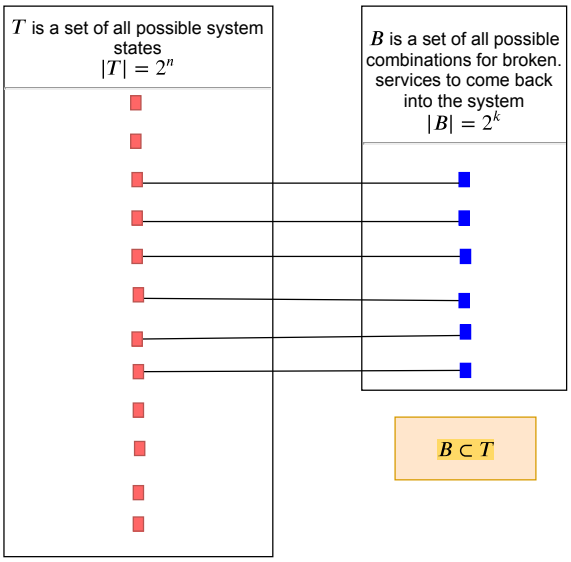


Fig. A.2 Subset relation representation between set of total possible system states with set of possible combinations of broken services getting back into the system.

2. The broken services come back into working state into the system all at once. From the possible combination of services coming back into the system, there will be only one favourable state that will bring both the services back into the system at the same time. The probability of such an event can be written as (equation A.8):

$$P(\text{Probability of broken services coming back into working state at the same time}) = \frac{1}{2^k} \tag{A.8}$$

Now, a system will be called resilient in both the possibilities written above. It will be more resilient if the second possibility happens. Analysis of equation A.8 suggests that if k is less which means correspondingly 2^k is less, then there will be more chances of the services coming back into the system at the same time. Which means that if less services are disrupted, the system will come back to it's original working state more quickly. This can be linked to the fault tolerance trait of a system, which is the working of the system in the event of services getting disrupted.

Without any doubt, there is a fine line between system resilience and fault tolerance, but a high fault tolerant system would ideally be more resilient. Having a distributed application architecture approach for design and development, and utilizing the computational resources along things to cloud continuum by means of fog computing adds to the fault tolerance trait of the developed solution, making it more fault tolerant and more resilient as well.

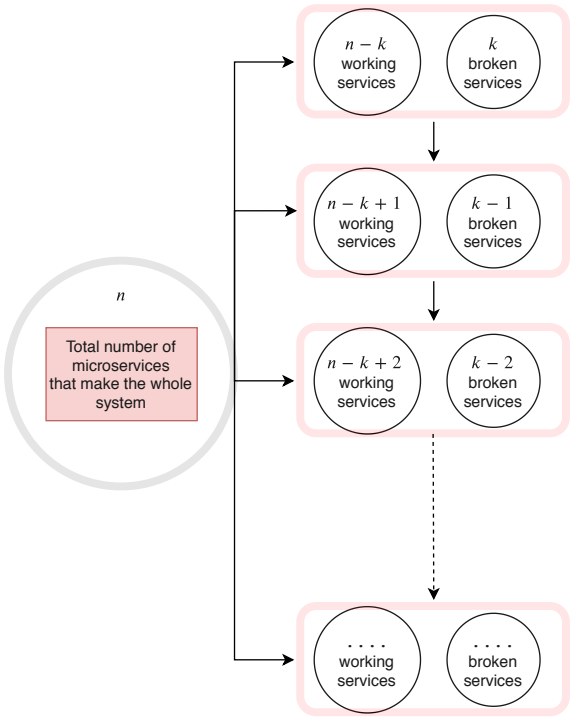


Fig. A.3 Diagrammatic representation of services getting broken and getting back into the pool of working service sequentially one after the another.

Resilient would be more related to how quickly the system can recover or come back to its fully working state; so the ideal quantity to measure will be time taken to recover, while in fault tolerance experiment it's the working of the system.

Appendix B

Using Cgroups in Benchmarking Experimentation

Below we provide a quick yet complete overview of control groups:

1. Install the control groups by — apt-get install cgroup-lite libcgroup1 cgroup-bin
2. cgroup-mount
   This command will mount all the resource folders such as cpu, memory, disk (blkio) and network in /sys/fs/cgroups/. Each resource folder represents a specific resource named as cpu, memory, disk (blkio) and network. You can set the required resources by setting the values in each folder. For our need we require just four folders.
3. Now create multiple sub-folders in the four folders above. These sub-folders will represent our resources. Simple mkdir command can be used.

   mkdir /sys/fs/cgroup/cpuset/1P1024M10D1N
   mkdir /sys/fs/cgroup/memory/1P1024M10D1N
   mkdir /sys/fs/cgroup/blkio/1P1024M10D1N
   mkdir /sys/fs/cgroup/netio/1P1024M10D1N

   The above four commands will create four sub-folders in each of the above resource folders. Similarly we can create as many folders as we want.
4. Now you need to assign the values to the control group. Like CPUs, Disk and memory etc, In the below example, I am assigning 1 physical CPU and 1024 MB (i.e., 1GB) main memory.

```
#!/bin/bash
echo "0" > /sys/fs/cgroup/cpuset/1P1024M10D1N /cpuset.cpus
echo "0" > /sys/fs/cgroup/cpuset/service/cpuset.mems
echo "1024M" > /sys/fs/cgroup/memory/1P1024M10D1N /memory.
   limit_in_bytes
```

Similarly, we can create as many folders as we want based on the lines in the Table 3.4 and assign the resources.

5. Now, we will assign all the processes running in the machine to the above group.

```
#!/bin/bash
for process in `ps -A | awk '!/PID/{print }'| awk '{print $1}'
    `; do cgclassify -g cpu,cpuset,blkio,memory:1V1024M10D1N
    --sticky $process; done
```

6. We can repeat 3, 4 and 5 for as many resource categories we have, which is limited to the maximum capacity of the machine.