DISSECT-CF-Fog: A Simulation Environment for Analysing the Cloud-to-Thing Continuum

PhD Thesis

András Márkus
Supervisor: Attila Kertész, PhD.

Doctoral School of Computer Science
Department of Software Engineering
Faculty of Science and Informatics
University of Szeged

Szeged
2022

Contents

List of Figures

List of Tables

1 Introduction

Internet of Things (IoT) is estimated to reach over 75 billion smart devices around the world by 2025 [68], which will dramatically increase the network traffic and the amount of data generated by them. The IoT paradigm composes sensors, actuators, and smart devices into a complex system through the Internet, which are utilised in various domains such as smart homes, smart cities, healthcare, agriculture, and transportation.

IoT systems often rely on Cloud Computing [57] solutions, because of their ubiquitous and theoretically infinite, elastic computing and storage resources. However, clouds are not always fitting for real-time IoT applications. If billions of IoT devices keep pushing their data to the cloud, it can be overloaded and the response time of the applications can dramatically increase. Fog Computing is derived from Cloud Computing to resolve the problems of increased latency, high density of smart devices, and overloaded communication channels, which is also known as the bottleneck-effect. The proximity of Fog Computing [15] nodes to end users usually ensures short latency values, however, these nodes are resource-constrained as well. Fog Computing can aid cloud nodes by introducing additional layers between the cloud and the IoT devices, where a certain part of the generated data can be processed faster [39].

IoT-Fog-Cloud systems have to deal with various challenges in order to ensure reliable IoT services [76]. Emerging issues of the IoT-to-Cloud continuum, such as connectivity problems of IoT devices, task offloading of computing nodes, billing and operation costs of substantial components, and resource provisioning are typically addressed by researchers. However, hiring physical machines from virtual server

parks fitting various IoT scenarios could be very expensive, and the investigation of IoT-enabled service compositions is not always possible with real cloud providers. As a result, in many cases simulators are applied to address such examinations with adequate results.

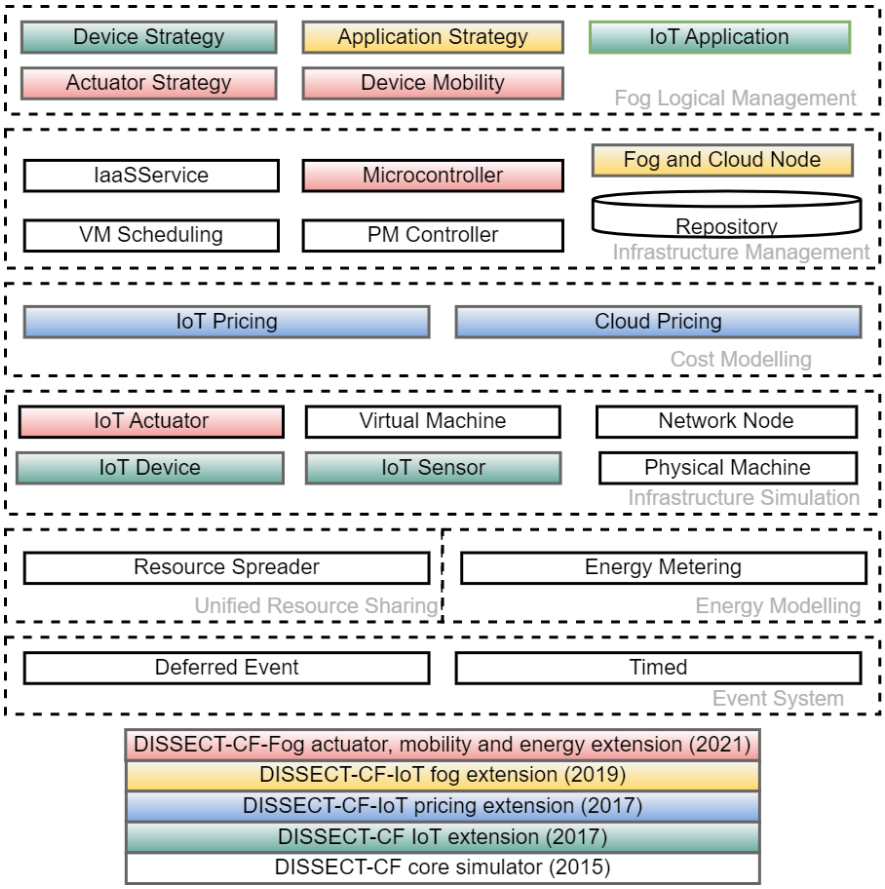


Figure 1.1: *The evolution of DISSECT-CF-Fog through its components*

The goal of my PhD research work was to address the challenges listed above. During the period of my doctoral studies, I had the opportunity to achieve my research goals by working with well-experienced researchers. This work started in 2016 as a BSc student, therefore, this dissertation concludes not just the last four, but the last six years of my research. During investigating the IoT-Cloud-Fog systems, which is often referred to as the Cloud-to-Thing Continuum [6], I have extended the DISSECT-CF discrete event IaaS (Infrastructure-as-a-Service) cloud simulator in four phases; this evolution towards DISSECT-CF-Fog is depicted in Figure 1.1. This dissertation goes through these phases by presenting my incremental research results as follows. **Chapter 2** presents a comprehensive overview of various IoT, cloud, and fog simulators in order to determine the key requirements of a compact and well-defined IoT-Fog-Cloud simulator. Furthermore, it presents an in-depth analysis and

a comparison of two major fog simulators. **Chapter 3** introduces the first two extension phases, namely the IoT and the pricing extension, exploiting a multi-cloud environment with resource allocation strategies. Finally, **Chapter 4** summarises the last two phases, where the simulator is extended towards modelling multi-layered fog systems with energy measurement and task allocation policies, and more detailed actuator and mobility events. The resulting DISSECT-CF-Fog simulator is open-source and available on GitHub[1].

1.1 Contributions

The ideas, figures, tables, and results included in this thesis were published in scientific papers (listed at the end of the thesis). The research results I achieved so far are organised into three theses, which are the following:

Thesis I. I analysed and classified numerous simulation approaches in terms of functionality, usability, maintainability, and code quality, in order to determine the most relevant properties for modelling IoT-Fog-Cloud systems. I also compared the two most prominent simulators in these fields, namely DISSECT-CF-Fog and iFogSim, with an in-depth performance analysis.

Thesis II. I designed a generic model of IoT systems and implemented it in the DISSECT-CF-IoT simulator. I developed a novel cost estimation extension using real cloud and IoT provider pricing schemes. I proposed various resource allocation strategies to reduce IoT application execution time and utilisation costs for multi-cloud environments. I also evaluated these strategies with a real-world meteorological use case.

Thesis III. I designed a generic model of Fog Computing and implemented it in the DISSECT-CF-Fog simulator to enable the modelling of the Cloud-to-Thing Continuum. I developed various task offloading policies for fog and cloud infrastructure management, to optimise IoT application makespan, utilisation costs, and energy consumption. I also proposed novel extensions to enable mobility and actuator behaviour analysis, and I evaluated these extensions with different smart system use cases.

Table 1.1 summarises the relationship between the thesis points and the corresponding publications and it also presents the citations received so far according to Google Scholar and MTMT.

[1]DISSECT-CF-Fog simulator (accessed in October, 2022): https://github.com/sed-inf-u-szeged/DISSECT-CF-Fog

Table 1.1: *Publications, theses and citations*

	Thesis I	Thesis II	Thesis III	Citations	
				Google Scholar	MTMT
[P3]	♦			48	36
[P7]	♦			4	4
[P11]	♦			1	2
[P1]		♦		13	11
[P2]		♦		10	7
[P5]		♦		10	7
[P6]		♦		3	4
[P10]		♦		5	3
[P4]			♦	1	1
[P8]			♦	-	-
[P9]			♦	1	-
Sum	3	5	3	96	75

1.2 Project Involvements

During the years of this research, I had the chance to be involved in numerous national and international projects and scholarships, the results of this thesis could not have been achieved without them. I am especially grateful for the support received from:

- the European COST program under action identifier CA19135 (CERCIRAS);
- the EU-supported Hungarian national grant GINOP-2.3.2-15-2016-00037 titled ”Internet of Living Things”;
- the Hungarian Scientific Research Fund under the grant number OTKA FK 131793;
- the Hungarian Government under the grant number EFOP-3.6.1-16-2016-00008;
- the UNKP-21-3 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund;
- the national project TKP2021-NVA-09 implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund;
- and the National Research, Development and Innovation Office within the framework of the Artificial Intelligence National Laboratory Programme.

2

A Survey and Taxonomy of IoT-Fog-Cloud Systems

2.1 Introduction

In the past decade, we experienced how rapidly distributed computing infrastructures evolve. The use of cloud technologies was almost unavoidable in 2010 for successful service providers of the future. Miniaturisation and improvements in battery lifetimes, which is another demand of the 21st century, have led to small computational devices that can interact and communicate among themselves and with the environment via the Internet giving birth to the Internet of Things (IoT) paradigm. As the number of things grows, the vast amount of data they produce requires the assistance of various services for storage, processing, and analysis. Cloud Computing became a good candidate to serve IoT applications, and their marriage created so-called smart systems [7].

Such IoT-Cloud systems can be utilised in many application areas ranging from local smart homes to mid-range smart cities, or wider smart regions. To cope with the possibly huge number of communicating entities, data management operations are better placed close to their origins, resulting in the better exploitation of the edge devices of the network. In the latest distributed computing paradigm called Fog Computing [39], groups of such edge nodes form a fog, where data processing and analysis can be performed with reduced service latency and improved service quality compared to remote cloud utilisation.

As a result, cloud and fog technologies can be used together to aid the data management needs of IoT environments, but their application gives birth to complex systems that still need a significant amount of research. It is obvious that significant investments, design and implementation tasks are required to create such IoT-Fog-Cloud systems in reality, therefore, it is inevitable to use simulations in the design, development, and operational phases of such establishments. This rationale has led many scientists to create simulators to investigate and analyse certain properties and processes of similar complex systems. There are already existing survey papers highlighting the basic capabilities of simulation tools modelling such complex systems, also comparing them by certain views, e.g. by Ragman et al. [58] or by Puliafito et al. [55]. Nevertheless, we believe that modelling IoT-Fog-Cloud architectures in such simulators is far from complete, and there is a need to gather and compare how key properties, especially of fogs, are represented in these works to trigger further research in this field.

The remaining sections of this chapter are organised as the following: Section 2.2 introduces related works proposing surveys in similar research fields and states our methodology for literature review. Section 2.3 is composed of five subsections, in which we give short introductions to the analysed simulators, define the taxonomy elements used for categorising them, provide classifications with comparison tables for three groups of simulators, and discuss the relevant findings of our classifications. Section 2.4 presents a comparison of the fog modelling capabilities of two distinguished simulators. Finally, the contributions are summarised in Section 2.5.

2.2 Related Works

The methodology for finding suitable works for our investigations was twofold. First, we looked for recently published surveys targeting Fog Computing and narrowed their scope to fog modelling and simulation. Hence, we filtered the group of their cited papers and kept the ones using simulators. In this way, we could build on their results, as well as go beyond their findings by further analysing these works.

Second, to extend the group of considered solutions, we performed a literature search with the following engines: Google Scholar, ResearchGate, Scopus, and Dimensions. We performed detailed searches for publications of the years 2015-2019 with the following keywords (for all fields): *Fog Computing* + *Simulation*. Table 2.1 shows the results of our literature search per publication year. We went through the search results and gathered only those works that contained solutions having open-source simulator implementations or containing novel approaches in the field of Cloud and Fog Computing. ResearchGate does not provide detailed information on search results, thus we omitted it from the comparison table. We also considered relevant solutions referenced in the processed articles, but we skipped those tools or

simulators, that were published before 2009.

Table 2.1: *Literature search results in June, 2019*

Year	Google Scholar	Scopus	Dimensions
2019	3,320	914	2,940
2018	6,140	1,560	5,447
2017	4,460	707	3,389
2016	2,870	310	2,100
2015	2,160	206	1,650
Sum	18,950	3,697	15,526

There are already a couple of survey papers addressing different aspects of Cloud Computing and Fog Computing supporting IoT systems. These works helped us to identify taxonomy categories for comparing our considered papers, i.e. solutions modelling Cloud and Fog Computing in simulated environments. Next, we summarise the most relevant surveys we found, then compare them with our research aims.

Rahman et al. [58] looked for available solutions for simulating Cloud Computing and modelling data centres and their networks. The authors investigated the capabilities of cloud simulators with the following categories: (i) graphical user interface, (ii) application model, (iii) communication model, (iv) energy model, (v) virtual machine (VM) support, (vi) SLA support, (vii) cost model. They investigated 15 simulators in detail, and also listed the applied programming language, the utilised platform, and the code availability of the simulators. This study is probably the closest to our work in terms of comparison methodology, but we focused more on the quality of the simulator software and broadened the research scope to IoT and fog areas.

Yousefpour et al. [77] presented a survey on Fog Computing and made comparisons with the following main points of view: computing paradigms such as cloud, fog, mobile, mobile-cloud, and edge, and frameworks and programming models for real applications, software, and tools for simulated applications.

Svorobej et al. [67] examined different fog and edge computing scenarios, and investigated Fog and Edge Computing with various aspects, such as (i) application level modelling, (ii) infrastructure and network level modelling, (iii) mobility and (iv) resource management, and (v) scalability, and made a comparison of seven available fog tools.

Hong and Varghese [26] summarised resource management approaches in Fog and Edge Computing focusing on their utilised infrastructure and algorithms, while Ghanbari et al. [22] investigated different resource allocation strategies for IoT systems. This second work classified the overviewed works into eight categories:

QoS-aware, context-aware, SLA-based, efficiency-aware, cost-aware, power-aware, utilisation-aware, and load balancing-aware resource allocation. We found this work particularly interesting, hence it opened the investigations into the field of IoT.

Puliafito et al. [55] investigated the benefits of applying Fog Computing techniques to support the needs of IoT services and devices. In their survey, they described the characteristics of fogs and introduced six IoT application groups exploiting fog capabilities. They also gathered fog hardware and software platforms supporting the needs of these IoT applications.

Table 2.2: *Comparison of related surveys according to their main contributions*

Survey	Year	Aim
Yousefpour et al. [77]	2018	Paradigms and research topics for fogs
Hong and Varghese [26]	2018	Resource management in the fog and edge
Rahman et al. [58]	2019	Simulating cloud infrastructure
Svorobej et al. [67]	2019	Fog and edge simulation challenges
Ghanbari et al. [22]	2019	Resource allocation methods for IoT
Puliafito et al. [55]	2019	Fog characteristics for IoT
Our survey	2019	Fog models and software quality of simulators

Table 2.2 summarises the relevant surveys representing the current state of the art close to our research aims. Though cloud solutions are dominating, some started to open investigations towards fog, edge, and IoT fields. To the best of our knowledge, there is no detailed study available targeting the fog modelling capabilities and the usability of simulation tools. In our work, we still do not neglect cloud and IoT solutions, hence they are tightly connected to fogs in most cases. We also investigate the software quality of the available simulators, which has a direct effect on the learning curve of utilisation and experiment reliability.

Our preliminary observation based on the literature analysis is that most simulation tools started to be developed as cloud simulators, and were later extended to model IoT and fog systems as well. However, occasionally these extensions were developed not by the original developer team, but other research groups. In the next section, we give a short introduction to the main goals and properties of these works and then compare them in a detailed taxonomy.

2.3 Simulation Environments of IoT-Fog-Cloud Systems

In this section, we present an introduction and detailed comparison of the currently available simulators in the fields of cloud, fog, and IoT. We considered the following properties as taxonomy categories for the investigation: (i) the topology and layers of

a simulator, (ii) the type of a simulator (i.e. a generic, event-driven or a specialised, network simulator), (iii) the date of the latest modification (to help with filtering outdated solutions), (iv) the applied cost model in a simulator (i.e. pre-defined, static or real provider pricing model), (v) geographic location management (mostly for fog placement optimisation), (vi) utilised sensor model, (vii) configurable network settings or protocols, (vii) VM management functionality, (viii) power usage or energy consumption calculations, and finally (ix) support for hierarchical organisation model (mostly in case of IoT and fog simulators).

2.3.1 Introduction of General Cloud Simulators

As we mentioned before, the life of most simulators we consider started as a cloud or network simulator. Some stayed at this level, and others were extended to support the analysis of fog, edge or IoT management. This means that we cannot avoid the investigation of cloud simulators.

One of the most referred to and widely used simulators is the CloudSim simulation toolkit [11], which is a popular solution for simulating cloud environments. In CloudSim, users define tasks by creating so-called cloudlets, which are processed by virtual machines running on cloud resources. This open-source, Java-based solution is built on SimJava [27]. CloudSim is a discrete event simulator, and its architecture has five layers (i.e. network, cloud resources, cloud services, VM services, and user interface structures). Virtual machines in CloudSim have three states, and users can configure only static usage costs for resources (i.e. memory, bandwidth, CPU, and storage usage). CloudSim also contains a power model, but it is only restricted to CPU energy consumption. Communication between components in the network is modelled with the bandwidth parameter and a delay value.

There are many extensions of CloudSim, focusing on mostly one aspect of Cloud Computing or providing the implementation for a missing or insufficient feature of the original software. Unfortunately, it seems that the developers found no need to collect all of the extensions into a single repository, and there are also many referred studies without publishing or citing available source code. Next, we introduce the ones we found relevant to our research.

CloudAnalyst [73] is one of the oldest extensions of CloudSim and contains important additions by implementing (i) application users, (ii) internet connections, (iii) simulations defined by time periods, (iv) service brokers. CDOSim [18] is an extension to simulate cloud deployment options for migration support. The simulator can measure the cost of deployment and its response time. The novelty of this work is to introduce MIPIPS, which is a fine-grained version of MIPS for low-level instruction definitions.

TeachCloud [29] is another extension for educational purposes. The main fea-

tures added in this version are: (i) a cloud workload generator, (ii) the MapReduce framework, and (iii) new cloud network models. EMUSIM [10] has the purpose to bring emulation and simulation into one tool to predict service behaviour, when the resource pool is changing. First, an emulation phase can be used to extract information on application behaviour, then in the second phase, this information is fed into a simulation model to arrive at more accurate methods for application simulation.

CEPSim [25] aim to simulate complex event processing (CEP) and stream processing systems in cloud environments. CEP is usually related to Big Data management, in this version CEP systems are represented by Directed Acyclic Graphs. Users can define how to process input streams with special queries that can run continuously in the simulation with user-defined runtime. CloudEval [72] is a CloudSim extension to support virtual machine consolidation with different migration algorithms. The authors have created a domain-specific language (DSL) for algorithm implementation for investigating specific VM migration strategies. The proposed DSL is based on the Groovy programming language, and it contains the following: descriptions for (i) the data centre and the VM scheduling strategies, (ii) evaluation metrics, and finally (iii) data collection and performance metrics.

CloudSimSDN simulator [63] is an extension to simulate Software Defined Networking (SDN) methods, which makes network elements (e.g. switches) dynamically programmable. Its new components (e.g. Link, Switch and Host) enable networking with virtual links in the physical topology. There is also a virtual topology layer in the top level of the CloudSimSDN architecture. The configuration of a simulation is based on JSON files which contain specifications for the nodes, policies and bandwidth values, and CSV files which contain the workload on the defined topology.

The ContainerCloudSim [52] package was created to respond to the growing trend of container technologies. It is an extension that enables running containers on top of VMs in a simulation. In this solution, the Container entity is hosted by a VM, and the container task has a very similar working as a Cloudlet. NetworkCloudSim [21] extends CloudSim with enhanced network options for modelling the interconnection of data centres. NetworkCloudlet is a novel component in this version, which can be used to simulate different task types (namely communication and computation tasks). Other extended components are the NetworkDataCenter and the NetworkHost (which can be managed by the former), these classes make it possible to simulate network traffic (using bandwidth and latency parameters) through switches. It also uses the ContainerCloudSim package listed earlier.

DynamicCloudSim [9] was created to simulate instability and dynamic changes in VM provisioning, which may occur in cases when the host machine serves more than one virtual machine at the same time. The newly introduced features are the following: (i) running different types of tasks (i.e. input-output, CPU, bandwidth-bound) on VMs, (ii) managing instability caused by hardware and network issues,

(iii) managing heterogeneity caused by different types of hosts, (iv) introducing VM performance changes and noise, and finally (v) defining failures during task execution. Finally, CloudSim Plus [20] is a redesigned and refactored version of CloudSim, which aims to provide easier usage and more reliable simulation.

Concerning solutions outside the CloudSim world, the DISSECT-CF cloud simulator [32] is an event-driven simulator written in Java. It can be used to model cloud Infrastructure-as-a-Service systems, having an energy consumption model both for physical and virtual machines with min, max and idle power states. Concerning resource management, there are four possible states of a physical machine, while virtual machines can be in 11 different states including migration. To model network communication between nodes, users can configure latency, as well as input, output and disk bandwidth. For further information, a recent study on its comparison to CloudSim is available in [41].

DCSim [70] is an event-driven simulator (using the Java programming language). It was developed for simulating virtualised resource management. The main ability of DCSim is to provide a VM sharing mechanism for VMs belonging to a single, multi-tiered application. Its architecture is composed of: DataCentre, Host, VMAllocation, and there are unique managers for networking, virtual machines, and power consumption. During simulation, the following values are measured: (i) dynamic VM allocation (migration), (ii) SLA violation, (iii) host operation hours and utilisation, (iv) power consumption and (v) simulation and algorithm running time.

GroudSim [51] aims to simulate grid and cloud systems in a scalable way. This Java-based software is a discrete-event simulator with the main ability to provide cost calculation, and simulations using the background-load of resources. The architecture is composed of Entity, Job, and Cost elements, and it contains a cost model using time units of CPU usage, or data usage (in GB).

GreenCloud [35] is an extension of NS-2 packet-level network simulator and is based on TCL scripts and C++. The main purpose of the simulator is to present energy-aware cloud data centre analysis including energy-usage measurements of the system components (i.e. servers, switches, links). Its energy consumption model includes PUE and DC infrastructure efficiency (both are represented in Watt). The architecture components can be servers and switches, and the tool is able to simulate TCP/IP protocols for networking.

ICanCloud [50] is an OMNET++ based simulator (programmed in C++), which aims to simulate cloud infrastructures. Its novelty is that it introduces a hypervisor component for managing cloud brokering policies. Its experiments are focused on the trade-offs between the cost and performance of a given application, which can be run with Amazon VM instance types only. The architecture has the following elements: cloud system, hypervisor, application repository, VM repository, and hardware models. It also contains a cost model, which follows the pay-as-you-go manner. The

virtual machine handling is quite simple, it can execute the jobs, but neglects cloud-specific network simulation. It uses the Inet framework (included in OMNET++) for TCP/UDP protocols.

Finally, SPECI [66] was developed with the aim to simulate cloud-scale data centres focusing on their performance and behaviour, which is based on a discrete-event simulator called DES Java.

Table 2.3: *Comparison of the examined cloud simulators with implementation-related properties*

Simulator	Core simulator	Last modified or published	Type
CloudSim	SimJava	2019	Event-driven
CDOSim	CloudSim	2012	Event-driven
NetworkCloudSim	CloudSim	2011	Event-driven
TeachCloud	CloudSim	2015	Event-driven
CloudAnalyst	CloudSim	2009	Event-driven
EMUSIM	CloudSim	2012	Event-driven
CEPSim	CloudSim	2016	Event-driven
CloudEval	CloudSim	2016	Event-driven
CloudSimSDN	CloudSim	2019	Event-driven
ContainerCloudSim	CloudSim	2019	Event-driven
DynamicCloudSim	CloudSim	2017	Event-driven
CloudSim Plus	CloudSim	2019	Event-driven
DISSECT-CF	-	2018	Event-driven
SPECI	DES Java	2009	Event-driven
DCSim	-	2014	Event-driven
GroudSim	-	2011	Event-driven
GreenCloud	NS-2	2011	Network
ICanCloud	OMNET++	2015	Network

Table 2.4: *Comparison of the examined cloud simulators with cloud modelling properties*

Simulator	Architecture	Cost model	Network model	VM management	Energy model
CloudSim	Network, Cloud Resources, Cloud Services, VM services and User Interface Structure	Static cost for physical resources: memory, storage, bandwidth, and CPU usage	Bandwidth and the delay value between the entities	3 states of VMs and it executes Cloudlets	Power model is based on max power and a constant (static power) values
CDOSim	Similar to parameters of CloudSim				
NetworkCloudSim	NetworkDataCenter, Switch and NetworkHost	Static cost for physical resources: memory, storage, bandwidth, and CPU usage	Simulate network traffic (bandwidth+latency) through switches	3 states of VMs and it executes Cloudlets	Power model is based on max power and a constant (static power) values
TeachCloud	Similar to parameters of CloudSim				
CloudAnalyst	Similar to parameters of CloudSim				
EMUSIM	Similar to parameters of CloudSim				
CEPSim	Similar to parameters of CloudSim				
CloudEval	Similar to parameters of CloudSim				
CloudSimSDN	Physical Topology (Link, Switch and Host and Cloud Data Center) and Virtual Topology	Static cost for physical resources: memory, storage, bandwidth, and CPU usage	Virtual link with bandwidth values	3 states of VMs and it executes Cloudlets	Power model is based on max power and a constant (static power) values
ContainerCloudSim	Similar to parameters of CloudSim				
DynamicCloudSim	Similar to parameters of CloudSim			Dynamic VM with task's length, I/O and bandwidth coefficient	Power model is based on max power and a constant (static power) values
CloudSim Plus	Similar to parameters of CloudSim				
DISSECT-CF	Event System, Unified resource sharing, Energy Modelling, Infrastructure Simulation and Infrastructure Management	N/A	Latency, input bandwidth, output bandwidth and disc bandwidth between nodes	11 states of Virtual Machine and it executes compute tasks	Energy model for PM and VM including min-, max- and idle power
SPECI	N/A				
DCSim	DataCentre, Host and VMAllocation	N/A	Bandwidth manager	simulated VM manager, and VM allocation	Power model includes idle- and max power
GroudSim	Entities (CloudSite,GridSite), Job, Cost	Cost calculation for job execution	Background load on resources	N/A	N/A
GreenCloud	Servers, Switches and Links	N/A	TCP/IP	N/A	PUE and DC infrastructure efficiency
ICanCloud	Cloud System, hypervisor, application repository, VMs repository hardware models	N/A	N/A	N/A	N/A

Table 2.5: *Comparison of the examined cloud simulators with software metrics*

Simulator	Language	Lines of code	Comments (%)	Duplication (%)	Files	Bugs	Vulnerabilities	Code smells
CloudSim	Java, XML	20,752	33.9	21.4	225	33	136	1.2k
CDOSim	Java	N/A						
NetworkCloudSim	Java							
TeachCloud	Java, XML, JSON	33,905	49.1	N/A	395	N/A		
CloudAnalyst	Java, HTML,XML	44,861	12.1	N/A	248	N/A		
EMUSIM	Java, XML, Python	1,665	3.33	N/A	21	N/A		
CEPSim	Java, Scala	5,875	20.5	N/A	82	N/A		
CloudEval	Java, Groovy	N/A						
CloudSimSDN	Java, XML	12,585	16.8	15.7	120	23	109	1.2k
ContainerCloudSim	Java	N/A						
DynamicCloudSim	Java, XML	16,778	31.5	14.2	173	65	428	754
CloudSim Plus	Java, XML	32,425	34.8	5.6	441	36	2	1.3k
DISSECT-CF	Java, XML	5,152	42.6	0.3	62	9	17	173
SPECI	Java	N/A						
DCSim	Java, Markdown	9,006	13.1	N/A	143	N/A		
GroudSim	Java	N/A						
GreenCloud	C++, TCL script	N/A						
ICanCloud	C++, HTML, XML, JS	2,901,003	4.1	N/A	12,463	N/A		

2.3.2 Introduction of IoT Simulators

Some of these cloud simulators followed the new trend represented by the Internet of Things. Such things are sensors and small devices that can be connected to the Internet, and used to monitor environments or gather special-purpose data. They are rarely used ”alone”, cloud services are generally needed to store and process their data. Once such data management is performed at the cloud network edge or close to the users on purpose, we arrive at the latest trend of Fog Computing. This reasoning led us to introduce the category of IoT simulators in our research. The revised cloud simulators had the aim to model entities of the Internet of Things world, which in some cases led to the introduction of fog features. Nevertheless, we can also find solutions focusing on IoT system operation, somewhat neglecting or hiding cloud and fog capabilities. In our view, IoT simulators belong to a separate category, hence IoT devices and sensors can be modelled as independent entities of data sources having special properties (e.g. data generation frequency, behaviour models, limited connectivity). In this section, we compare works focusing on IoT management.

SysML4IoT [14] is an extension of SysML and it is designed for model-driven development for IoT systems. In SysML4IoT an IoT system has two main compo-

nent groups: devices (Tag, Sensor, Actuator), and services (Human, Digital Artifact), and this approach follows the publish/subscribe pattern to specify sensor behaviour. With this tool a high-level model can be designed of an IoT system that can later be transformed into a source code, hence an implementation of a system.

CrowdSenSim [19] is a simulator designed for mobile crowd-sensing. The main features are the following: (i) users can scan the layout of cities, (ii) can manage user mobility and (iii) perform energy measurements for sensing tasks. The modules or layers are the following: User Mobility, City Layout, and CrowdSensing module. For managing mobility, the latitude, longitude, and altitude of a device are considered, and it may also use real sensors (e.g. accelerometer, temperature and pressure). This simulator contains a power model using idle, transmission and reception modes, and the cost of the sensing is also measured.

The DPWSim [24] simulation toolkit helps to develop and test IoT applications with web services using the DPWS standard without any physical devices. DPWS works with publish/subscribe mechanisms on the architecture based on spaces, devices, operations and events. The main abilities of the DPWSim are the following: (i) Platform Independence due to JVM, (ii) virtual device management for DPWS and it supports SOAP and HTTP protocols.

SimIoT [65] is an extension of SimIC with an IoT layer, and the authors demonstrate its utilisation with an IoT-based healthcare application use case. This extension provides an extra layer for communication between the entities. Its architecture has three main layers: User level (device-sensor), SimIoT level (communication broker), and SimIC level (cloud entities). The virtual machines are used to execute tasks, and there is a bandwidth constraint for governing the network load.

A few studies mention the use of special software to simulate or manage IoT systems. Angelakis et al. [3] used the Mixed Integer Linear Program in Matlab to model resource allocation problems in heterogeneous IoT networks. Simulink [37] is also a Matlab-based solution that provides secure model-based design for IoT system analysis. Thomas and Irvine [69] presented an LTE approach for IoT sensor networks. They performed experiments with the OMNET++ network simulator to investigate how many sensor nodes can be transmitted per resource block.

The SmartSim tool [13] was designed for energy-metering in IoT smart homes. This simulation tool was developed in Python, and it is able to generate smart device energy traces based on energy models (with frequency, duration, time and activity). There is a semi-simulated solution for modelling IoT, fog, and cloud systems called MobIoTSim [34], which is an Android-based software for IoT device simulation and management. It was designed with the aim to avoid expensive device and sensor purchases for developing and evaluating IoT applications. By executing simulations, the mimicked devices can connect to real cloud providers (e.g. IBM Bluemix), and communicate over the network using MQTT and HTTP protocols. MobIoTSim has

the following layers: sensor, device, application, and cloud.

IOTSim [78] is a CloudSim extension that supports Internet of Things and Big Data simulation. IoT has appeared in the CloudSim model as a three-layer architecture: perception, network, and application layer, with the following details: CloudSim Simulation Layer, Storage Layer, Big Data Processing Layer, and User Code Layer. The aim of this work was to simulate a MapReduce approach for Big Data processing, hence the novelty of this work is the MapReduce function implemented by the MapCloudlet and ReduceCloudlet entities. Unfortunately, no detailed information can be found on sensors or actuators, and there was no available source code cited.

Finally, DISSECT-CF-IoT [P1] is an extension of the DISSECT-CF cloud simulator. The novelty of this extension is the sensor model, which contains detailed configuration options (e.g. measurement delay, data generation frequency, and generated file size parameters), detailed network configurations for IoT devices, and different approaches for multi-cloud management strategies for IoT applications. It also contains extensible pricing models of four real providers (Amazon, Azure, IBM Bluemix, and Oracle), both for IoT and cloud-side costs. To enable large-scale simulations, it uses different XML description files for easy configuration management.

Table 2.6: *Comparison of the examined IoT simulators with implementation-related properties*

Simulator	Core simulator	Last modified/ published	Type
SysML4IoT	-	2016	-
CrowdSenSim v1.0.0	-	2017	Event-driven
DPWSim	-	2014	Service-messaging events
SimIoT	SimIC	2014	N/A
SmartSim	-	2016	N/A
IOTSim	CloudSim	2016	Event-driven
MobIoTSim	Android	2019	Semi-simulated
DISSECT-CF-IoT	DISSECT-CF	2019	Event-driven

Table 2.7: *Comparison of the examined IoT simulators with IoT modelling properties*

Simulator	Architecture	Cost model	Geolocation	Sensor model	Network model	VM management	Energy model
SysML4IoT	N/A						
CrowdSenSim v1.0.0	User Mobility, City Layout and CrowdSensing module	Only energy cost of the sensing	Latitude, longitude and altitude	Real sensors	N/A		power in idle, transmission and reception mode
DPWSim	Spaces, Devices, operations and Events	N/A	N/A	N/A	SOAP and HTTP protocols	N/A	N/A
SimIoT	User level (device-sensor), SimIoT level(communication broker) SimIC level (Cloud Entities)	N/A	N/A	N/A	Bandwidth constraint	VM for execute task	N/A
SmartSim	N/A						
IOTSim	CloudSim Simulation, Storage, Big Data Processing, User Code Layer	Similar to parameters of CloudSim					
MobIoTSim	Sensor, Device, Application and Cloud layer	N/A	N/A	N/A	MQTT	N/A	N/A
DISSECT-CF-IoT	Sensor, Smart Device, and Cloud	Dynamic IoT and cloud-side costs	N/A	Delay, frequency and file size	Latency, input bandwidth, output bandwidth and disc bandwidth between nodes for devices and nodes	11 states of Virtual Machine and it executes compute tasks	Energy model for PM and VM including min-, max- and idle power

Table 2.8: *Comparison of the examined IoT simulators with software metrics*

Simulator	Language	Lines of code	Comments (%)	Duplication (%)	Files	Bugs	Vulnerabilities	Code smells
SysML4IoT	N/A							
CrowdSenSim v1.0.0	HTML, JS, C++, Python	44,437	2.8	N/A	346	N/A		
DPWSim	Java,HTML	55,346	51.2		551			
SimIoT	Java	N/A			N/A			
SmartSim	Python	946	12.7		13			
IOTSim	Java	N/A			N/A			
MobIoTSim	Java,XML	5,490	4.6		75			
DISSECT-CF-IoT	Java,XML	7,160	37.1	0.3	91	23	90	306

2.3.3 Introduction of Fog Simulators

Finally, we arrive at the latest category of fog simulators responding to the current trend called Fog Computing. In this category, we focus on key properties required to model a Fog Computing environment. In their development we can also identify the close relation to clouds: most of them are extensions of cloud or IoT simulators. Nevertheless, they follow different architectural models: some have centralised, while some more decentralised, peer-to-peer communication schemes. As a result, understanding the model elements, functions, and implementation details can be very hard and time-consuming; that is one of the issues our survey aims to relax.

Brogi et al. [8] presented a novel cost model for deploying applications on a fog infrastructure. The approach is based on a simulation prototype called FogTorchΠ, which determines a deployment on fog according to actual resource consumption and cost needs. To calculate these metrics, the simulator uses Monte Carlo methods. The main capabilities of this simulator are the monthly cost calculation extended with subscription/data transfer cost of IoT devices (using bandwidth and latency parameters), and it takes into account geolocation information. The cost model differentiates two methods: (i) to choose one pre-defined virtual machine with the given costs, or (ii) to build the necessary units of the CPU cores, RAM, and HDD for a virtual machine. The connections of fog nodes and clouds have different types, and both vertical and horizontal hierarchies are possible. The realisation of an IoT-Fog-Cloud architecture follows the 3-tier model sensor-fog-cloud, but it lacks some important features of cloud/fog simulation (e.g. virtual machine simulation and management). This tool focuses on providing a fog-searching algorithm for the given parameters. FogDirSim is a closely related simulation tool from the same authors, which provides compatibility with CISCO FogDirector across REST services to securely manage IoT applications on fog architectures.

Abbas et al. [1] presented a work using the OPNET network simulator and aims

to present the fog security service for end-to-end security between the fog layer and IoT devices. They represent the difference between the cloud-fog-device layer architecture and the fog-device architecture with a decentralised approach. The authors describe an encryption solution for the fog layer and its devices in a mixed real-life system and simulation possibilities. First, they run a scenario with real devices (e.g. iPhone, Samsung Galaxy) to get some benchmark values, then they use the OPNET network simulator parameterised with the measured values to get information about different traffic loads.

Tychalas and Karatza proposed PDES [71] for Fog Computing, the Parallel Discrete Event Simulation implemented in the C programming language. They defined a system with the following parameters: a cloud of 128 VMs, a cluster of 32 raspberries, a cluster of 64 PCs and 64 smartphones. If a task arrives to the system, it can be stored in a resource queue. The tasks are independent, and they prefer the shortest queue with the lowest load for placing a task. There are three thresholds in the system for task allocation (representing the cloud, raspberry and smartphone categories). It seems that there is no possibility for the detailed and dynamic configuration of simulated system components, the provided solution focuses on analysing task scheduling algorithms to decrease the response time.

The FogNetSim++ [56] is built on the OMNeT++ discrete event simulator, which focuses on network simulation. This simulator was designed to provide configuration options to handle fog networks (e.g. fog node scheduling algorithms and devices hand-over). The implementation programming language is C++. The main capabilities of this simulator are the handling of communication protocols such as MQTT or CoAP, and the different mobility models such as LinearMobility or TractorMobility.

Edge-Fog [44] is a Python-based simulator, in which a fog layer is represented by networking devices (e.g. routers and switches). The authors use this tool to present their LPCF algorithm (i.e. least processing cost first), which orders tasks to available nodes by minimising processing time and network costs. This approach is decentralised, thus the edge layer has a device-to-device connection.

The Yet Another Fog Simulator (YAFS) [36] is proposed to simulate application deployment on a fog infrastructure. The main capabilities of the simulator are the following: (i) dynamic application module allocation, (ii) network failures, and (iii) user mobility. YAFS is a Python-based discrete event simulator and provides JSON-based scenario definition. The simulation results contain information such as network utilisation, response time, and network delay.

EdgeNetworkCloudSim [61] is an extension of CloudSim that supports the edge computing paradigm focusing on consolidation and orchestration on the edge (mostly mobile) devices. EdgeNetworkCloudSim is based on NetworkCloudSim, and the extensions are the following: EdgeService models the service chain, EdgeDatacenterBroker communicates with a user, and EdgeVms helps to define a service app.

The PureEdgeSim[2] tool was proposed to design and model cloud, fog, and edge applications focusing on the high scalability of devices and heterogeneous systems. PureEdgeSim is based on CloudSim Plus, and it uses XML descriptions for the simulation, where the users can configure the data and parameters of geographical location, energy model and virtual machine settings.

One of the most referred fog simulators is iFogSim [23], which is also based on CloudSim. iFogSim can be used to simulate real systems and follows the sensing, processing and actuating model, therefore, the components are separated into these three categories. The main physical components are the following: (i) fog devices (including cloud resources, fog resources, smart devices) with the possibility to configure CPU, RAM, MIPS, uplink- and downlink bandwidth, busy and idle power values, (ii) actuators with geographic location and reference to the gateway connection, (iii) sensors that generate data in the form of a tuple representing information. The main logical components aim to model a distributed application: the (i) AppModule is a processing element of iFogSim, and the (ii) AppEdge realises the logical data flow between the virtual machines. The main management components are as follows: the (i) Module Mapping searches for a fog device to serve a virtual machine, if no such device is found, the request is sent to an upper-tier object, and the (ii) Controller launches the application on a fog device. For simulating fog systems, first, we have to define the physical components, then the logical components, and finally the controller entity. Although numerous articles and online source codes are available for the usage of this simulator, based on our experiments suggest that there is a lack of source code comments for many methods, classes, and variables. As a result, application modelling with this tool requires a relatively long learning curve, and its operations take valuable time to understand. After its appearance, it has been used for many research works and various experiments. For example, in [47] a smart city network architecture is presented for Fog Computing called FOCAN as a case study, and in [5] the authors proposed to combine Fog and the Internet of Everything into the so-called Fog of Everything paradigm.

There are also more advanced extensions of iFogSim. MyiFogSim [38] was designed to manage virtual machine migration for mobile users. The main capability of this simulator is the modelling of user mobility and its connection with the VM migration policy. The evaluation contains a comparison to a simulation without VM migration. Another extension is the iFogSimWithDataPlacement [45] tool, which proposed a way for data management investigation on how data is stored in a fog system. It also considers latency, network utilisation, and energy consumption for data placement. The authors presented a parallel Floyd-Warshall algorithm to define the shortest distance between the nodes for optimal data placement.

[2]PureEdgeSim simulator online (accessed in June, 2019): https://github.com/CharafeddineMechalikh/PureEdgeSim

EdgeCloudSim [64] is another CloudSim extension that is available on GitHub[3]. The main capabilities of this simulator are the network modelling extension for WLAN, WAN, and the device mobility. They aimed to respond to the disadvantage of iFogSim's simple network model, which ignores network load and does not provide content mobility.

SpanEdge [60] is another decentralised tool related to edge computing, aiming to model data stream processing. Developers can build up a geographically distributed network by installing the parts of a stream processing application near the data source for latency and bandwidth reduction. We can also find simulation solutions in the field of Fog Computing exploiting the use of Matlab, Docker or other real service APIs. Zhang et al. [79] proposed to investigate simulated resource allocation in a 3-tier fog network by using the MatLab framework. They aim to provide management functionalities for data service operators, data service subscribers, and fog nodes.

DockerSim [49] aims to support the analysis of container-based SaaS systems in simulated environments. It is based on the OMNET++ and iCanCloud network simulators to model container behaviour, network, protocol and OS process scheduling behaviour. EmuFog [43] is an extensible emulation framework for fog infrastructures, and also a useful tool for emulating real applications. After designing a network topology, EmuFog enables connecting fog nodes in the topology and to run Docker applications on it.

DISSECT-CF-Fog [P7] is a direct extension of DISSECT-CF-IoT, written in Java programming language. The purpose of this simulator extension is to model fog devices and nodes, and by building on the core DISSECT-CF simulator and on DISSECT-CF-IoT extension functions, it is able to model IoT-Fog-Cloud systems. The main benefit of this fog extension is the possibility of detailed configuration settings through XML configuration files, and it requires only minimal programming knowledge to define additional scenarios. DISSECT-CF-Fog contains its own simulation time unit for the general and time-independent simulations. The network operations, such as bandwidth, latency simulations, and file transfers between the physical components are supported by its core simulator. To create a physical topology, any horizontal and vertical connections are allowed. The IoT layer provides the management of smart devices and those sensors with the ability to simulate sensor measurement time and data generation frequency with size configuration. Its application management layer handles the VMs and pairs the compute tasks (generated based on the received amount of data) to the VMs. The cost module is responsible for evaluating pricing methods, capable of calculating both dynamic cloud and IoT-side costs based on real provider schemes. Finally, the logical dataflow is defined by the physical

[3]EdgeCloudSim simulator online (accessed in June, 2019): https://github.com/CagataySonmez/EdgeCloudSim

topology by default for simplicity.

Table 2.9: *Comparison of the examined fog simulators with implementation-related properties*

Simulator	Core simulator	Last modified/ published	Type
FogTorchΠ	-	2018	N/A
FogDirSim	-	2018	N/A
OPNET	-	2019	Network
PDES	-	2018	Event-driven
FogNetSim++	OMNET++	2018	Network
Edge-Fog	-	2017	N/A
YAFS	-	2019	Event-driven
EdgeNetworkCloudSim	NetworkCloudSim	2017	Event-driven
PureEdgeSim	CloudSim Plus	2019	Event-driven
iFogSim	CloudSim	2017	Event-driven
MyiFogSim	iFogSim	2017	Event-driven
iFogSimWithDataPlacement	iFogSim	2018	Event-driven
EdgeCloudSim	CloudSim	2019	Event-driven
SpanEdge	-	2016	N/A
Zhang et al. - Matlab	-	2017	N/A
DockerSim	iCanCloud	2017	Network
EmuFog	-	2019	Emulator
DISSECT-CF-Fog	DISSECT-CF-IoT	2019	Event-driven

Table 2.10: *Comparison of the examined fog simulators with fog modelling properties*

Simulator	Architecture	Cost model	Geolocation	Sensor model	Network model	Energy measurement	
FogTorchII	Cloud Data Centre, Fog Node and Thing	N/A	Coordinates	Coordinates	Latency, bandwidth	N/A	N/A
FogDirSim	N/A						
OPNET	Device, Fog and Cloud	N/A	Benchmarked values	N/A	Benchmarked values	N/A	N/A
PDES	N/A						
FogNetSim++	Devices, Fog nodes, Broker node and Base station	Pay-as-you-go, subscription (monthly, etc.), pay-for-resources and hybrid model	Regions	Sensor node is data generator and user node generate or receive data, wire and wireless nodes	Execution delay, packet error rate, handovers, latency	N/A	Energy model for devices and fog nodes based the task computed
Edge-Fog	Edge and Fog and Data Store layer	N/A					
YAFS	Cloud, Fog, Sensor and Actuator	Cost of execution in cloud	Coordinates	Instructions, bytes	Bandwidth and link propagation	N/A	Power in watt
EdgeNetworkCloudSim	EdgeService, EdgeDatacenterBroker˜ and EdgeVms	Similar to parameters of CloudSim			Simulate network traffic (bandwidth+latency) through switch	Similar to parameters of CloudSim	
PureEdgeSim	Cloud, Fog and Edge	N/A	Coordinates	N/A	Network usage (LAN and WAN bandwidth), latency latency requirement allocated bandwidth for each task	Task file size, tasks CPU utilisation 3 state of VMs	Energy consumption, battery capacity, idle and max consumption
iFogSim	Sensors, Actuators, Fog devices and Data Centers	Static cost for physical resources: memory, storage, bandwidth and CPU usage	Latitude, longitude	Output size, latency, CPU usage length, network usage length	Uplink bandwidth, downlink bandwidth, and uplink latency	3 state of VMs executes Tuples	Power model is based on max power and a constant static power) values
MyiFogSim	Mobile sensors, Mobile actuators, Mobile˜ devices, and Data centre	Static cost for physical resources: memory, storage, bandwidth and CPU usage	Coordinates	Similar to parameters of iFogSim			
iFogSimWithDataPlacement	New components: DataPlacement, Infrastructure Partition, WorkLoad Repartition	Similar to parameters of iFogSim					
EdgeCloudSim	CloudSim + Network, Edge Server and Mobile Client	Static cost for physical resources: memory, storage, bandwidth and CPU usage	Coordinates	N/A	Transmission delay, WLAN/WAN, upload/download data	Similar to parameters of CloudSim	
SpanEdge	Edge Data Centre, ˜ Central Data Centre with master-worker connection	N/A	N/A	Size in Bytes, message Count	Latency	N/A	N/A
Matlab (Zhang et al.)	N/A						
DockerSim							
EmuFog							
DISSECT-CF-Fog	Sensor, Smart Device, Fog Node and Cloud	Dynamic IoT and cloud-side costs	Coordinates	Delay, frequency and file size	Latency, input bandwidth, output bandwidth and disc bandwidth between nodes for devices and nodes	11 states of Virtual Machine and it executes compute tasks	Energy model for PM and VM including min-, max- and idle power

Table 2.11: *Comparison of the examined fog simulators with software metrics*

Simulator	Language	Lines of code	Comments (%)	Duplication (%)	Files	Bugs	Vulnerabilities	Code smells
FogTorchΠ	Java, XML	2,748	15.9	8.3	39	21	31	308
FogDirSim	Python, YAML	5,641	1.4	N/A	84	N/A		
OPNET	N/A				N/A			
PDES	C	N/A						
FogNetSim++	C++	20,199	5.7		59			
Edge-Fog	Python	887	17.2		66			
YAFS	Python, JS, HTML, JSON	31,597	22.0		208			
EdgeNetworkCloudSim	Java, HTML	113,654	27.5		571			
PureEdgeSim	Java, XML	3,308	12.2	4.3	30	18	101	301
iFogSim	Java, XML	27,754	25.3	24.3	290	124	248	1.5k
MyiFogSim	Java, XML	32,723	23.2	23.5	328	174	275	2k
iFogSimWithDataPlacement	Java, Protocol Buffers	212,780	7.6	N/A	2,313	N/A		
EdgeCloudSim	Java, XML	6,232	14.3	29.7	54	14	22	496
SpanEdge	Java, XML	1,417	10.3	34.1	17	9	11	232
Matlab (Zhang et al.)	N/A			N/A	N/A	N/A		
DockerSim	C++, INI	48,118	22.7		336			
EmuFog	Java	2,570	77.6		52			
DISSECT-CF-Fog	Java, XML	9,870	33.3	2.0	118	31	192	482

2.3.4 Detailed Taxonomy for Fog Modelling in Simulators

To compare the works we introduced in the previous subsections we define 10 taxonomy categories. These categories appear in the comparison tables we placed next to the introduction of the considered simulators: Table 2.3, Table 2.4 and Table 2.5 are used to summarise cloud simulators; Table 2.6, Table 2.7 and Table 2.8 are used for IoT simulator comparison; and finally Table 2.9, Table 2.10 and Table 2.11 depicts and compares properties of fog simulators. Next, we define our taxonomy elements, then provide a discussion for the mapped survey papers:

- Simulation type:

  Simulation in general has a long history of research, and many classification schemes are available. Roth [59] stated that there are three major types of simulation models: discrete, continuous, and combined. For dynamic systems, discrete event simulation models are the most widespread. In order to categorise the overviewed solutions, we define the following types of simulators: (i) *Network Simulators* aim to simulate the network connections and data transfers between the nodes. They are useful for modelling low-level interactions in systems, but their disadvantage is that it is hard to create higher-level abstract components (i.e. cloud or fog resources, IoT sensors) to build up complex systems (from their building blocks, i.e. network entities). Generally, in these cases, the simulation time may increase significantly. The other type category

is the general, (ii) *Event-driven Simulators*, which use the discrete event simulation model, where the inner working of the systems can be modelled by specific time moments (i.e. events). These events are usually mapped to system states, and the state transitions to certain component operations. The disadvantage of these simulators is the lack of built-in network operations and properties, however, one may choose the level of abstraction more easily, which is an advantage – as we have seen in DISSECT-CF or NetworkCloudSim to take into account the network traffic during the generation of virtual machines, communication or data forwarding. In Figure 2.1, we can see the ratio of the defined simulator types for the considered works. More than half of the investigated simulators (with 63.6%) fall in the event-based category to simulate complex systems composed of cloud, IoT or fog elements.

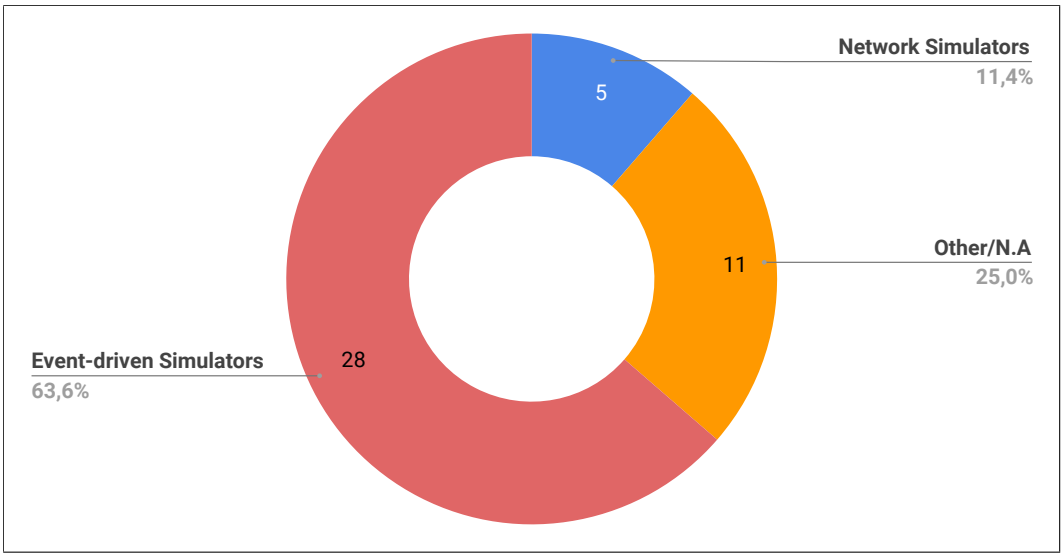


Figure 2.1: *The ratio of the simulator types*

- Implementation:

  The next category aims to reflect the development details of the considered simulators. Many of them have a single developer or a small group of contributors, only some have bigger developer communities (CloudSim or OMNET++).

  We have seen in the previous subsections that some solutions became very popular, and influenced other researchers to extend the core tools with additional functionalities. Figure 2.2 presents a graph highlighting the connections among the overviewed simulators based on the realised extensions. The bottom circle represents the core or base simulators, their extensions within the same system categories are placed on top of them, while the arrows lead to extensions for solutions modelling other systems as well. This graph shows that many variations

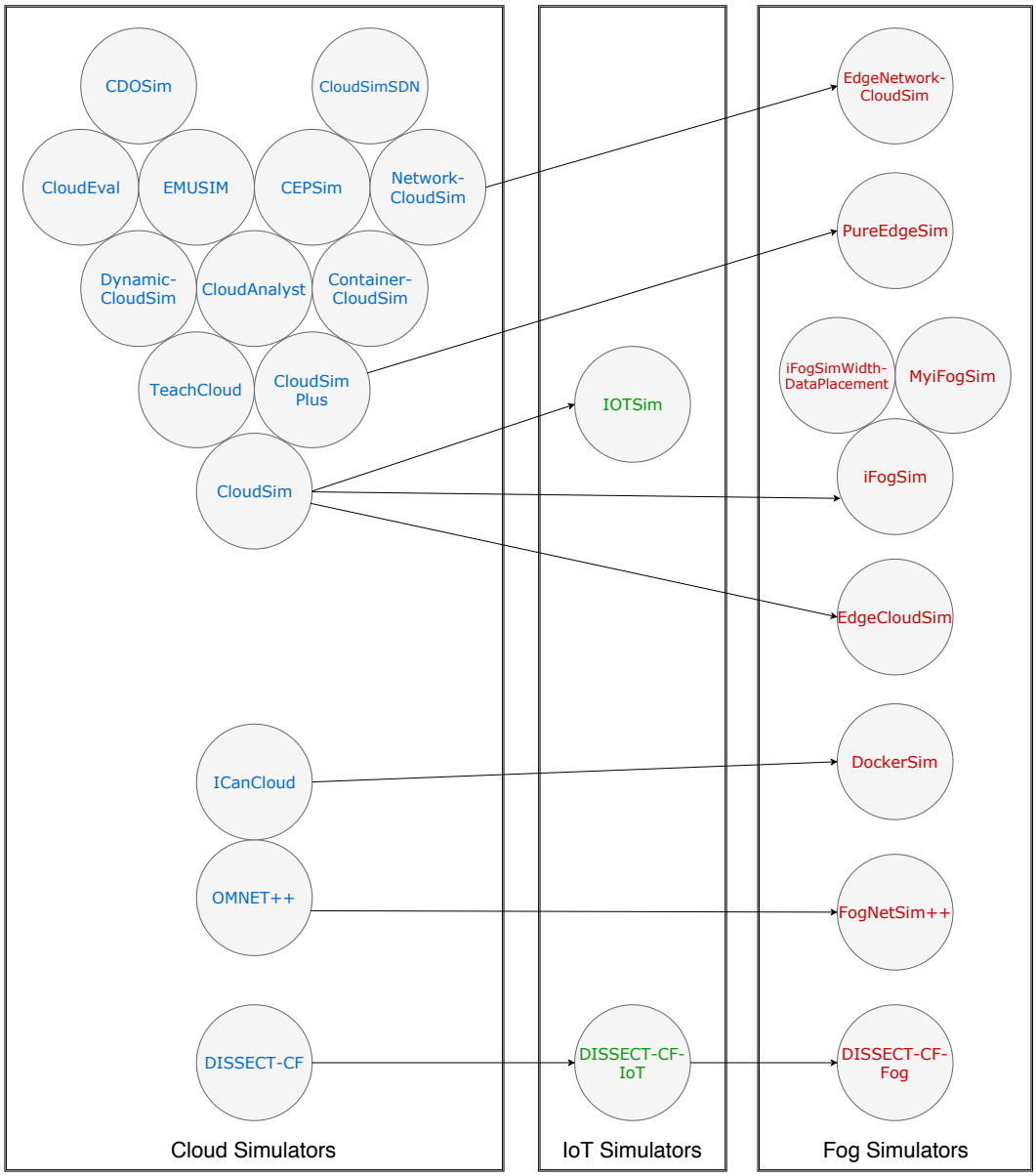


Figure 2.2: *Visualised relationships between the examined cloud, IoT, and fog simulators*

of a base simulator exist, and we also know that a concrete simulator has many development versions that may have different features. This fact makes it very hard for researchers to choose the right version for their investigations, and for developers to create an improved solution of different versions of the same simulator. Our taxonomy aims to reveal some implementation details later, in this regard.

- Publication date:

In the last decades we have seen the evolution of distributed systems: cloud systems matured around 2010, then various things started to appear to form IoT systems to generate data for clouds, and finally fog nodes were created to improve application execution quality of cloud services. We believe that the publication and development dates can help us to place the considered simulators in this evolution time frame. The publication date of a solution determines the technological novelties and capabilities its model is based on. It is true that a well-maintained and updated simulator can follow the latest trends, but its software can wear out in a few years due to corrections and extensions. The comparison tables reveal that CloudSim and DISSECT-CF were born quite early, and they are still under development to respond to the ongoing technological changes. On the other hand, old tools without improvements cannot fulfil recent research needs.

- Cost model:

  To predict the costs of certain operations in a complex system is an important and useful feature for researchers. With the advent of commercial solutions in these distributed systems, it became inevitable to provide simulated cost calculations for users planning to enter this market. Though commercial solutions for Fog Computing are still in their infancy, we can find various cost models for clouds and IoT in the considered simulators.

- Geolocation:

  One of the goals of Fog Computing is to reduce data transfer and service response times, which require the use of geographical information of system elements (e.g. sensors, nodes or users). In most cases, simulators having this property use two different representations: storing the (i) X and Y coordinates of an element in a system or the (ii) longitude and latitude values for more specific distance calculations. We applied this taxonomy category only for IoT and fog simulators. Mobility is a closely related property to geolocation, solutions offering this feature enable dynamic changes in the location of certain system elements.

- Sensor model:

  Sensors are key elements of IoT systems, and in many cases, they appear in fog environments as well. Therefore, it is important to know how sensors (or devices or things) are represented in certain simulators. In general, they should provide interfaces to discover, connect and monitor, they have certain data generation frequency, possibly data storing or queuing properties. Their model may also reflect behavioural information, such as actuation, failures or latencies.

- Network model:

  One of the crucial points of a simulator for distributed systems is how it handles the network operations, especially the representation of bandwidth and latency. The corresponding configuration settings, and the fine- or coarse-grained networking functions usually have a significant impact on the simulation time and accuracy of a simulator. Fine-grained models can support low-level protocol representations (e.g. HTTP or MQTT communication) to provide realistic simulations.

- VM management:

  Representing virtual machines and their management functions are some of the basic properties of modelling clouds and fogs. They determine the configuration means of VMs in the simulated environments, which can highly affect the usability of the simulator. The corresponding features of this category are: migration, task execution, network load, and background workload. Some fog solutions neglect this category to focus more on physical fog node management (e.g. FogTorchΠ or YAFS).

- Energy model:

  Physical elements of real-life distributed systems consume energy and affect carbon emissions. In order to develop solutions reducing these values, simulators should provide energy metering and power usage predictions. In general, three options are used for energy models: idle, min, and max consumption.

- Source code metrics:

  As a final taxonomy element we chose source code metrics to represent and compare the software quality of the considered simulators. In general, bad software quality makes it hard to read, understand, and reuse the source code for extensions, and it also negatively affects simulation time and accuracy. In this work, we used the static code analyser called SonarQube[4] to calculate certain quality metrics. This tool is able to measure various quality features of the source code of a software. In cases we could not apply SonarQube, we used an even simpler tool called CLOC[5] that can handle almost any type of programming language, but provides less information. Certainly, we could analyse only simulators with published, open-source code. We considered the following metrics for our investigation:

  – Language: the applied programming language to implement the simulator.

[4]SonarQube online (accessed in May, 2019): https://www.sonarqube.org
[5]CLOC online (accessed in May, 2019): http://cloc.sourceforge.net

We also highlighted additional languages where applicable, e.g. the ones used for serialisation.

- Lines of code: this metric represents the number of lines in the source code. Note that blank lines are not counted.
- Comments: this metric counts the ratio (in %) of comment lines to the total lines of code (i.e. blank lines plus lines of code).
- Duplication: it denotes the ratio of the code duplication to the whole source code.
- Files: this metric tells us the number of files that comprise the software.
- Bugs: it shows the number of bugs in the software, where bugs represent wrong language constructions (e.g. missing operand casting), according to the definition of SonarQube.
- Vulnerabilities: this metric tells us the number of vulnerabilities in the software, which are possible security issues (e.g. public member notation in a class) in the SonarQube terminology.
- Code smells: it counts the number of code smells, which are code blocks where modification and understanding could be time-consuming (e.g. empty statements), based on the definition of SonarQube.

In general, when a tool has acceptable quality metrics, this also has a positive effect on the application of the simulator. Concerning the evaluation of the metrics, we can state that higher values for code duplication complicate the readability of the source code, and the bigger ratio of comment lines helps researchers understand and reuse the code. Needless to say, the researchers would prefer fewer bugs, vulnerabilities, and code smells when deciding to use a simulator for an investigation. Concerning the implementation of the reviewed works, Figure 2.3 shows that more than 70% of the investigated simulators are written in Java, which is a platform-independent programming language.

2.3.5 Discussion and Future Research Challenges

In this subsection, we provide further discussions on the comparison of the analysed simulators. In the previous subsections, we provided short introductions to the main properties of the overviewed works, defined the taxonomy elements used for categorising them, and provided classifications with comparison tables for three groups of simulators, namely: cloud, IoT, and fog simulators. Though the tables provide

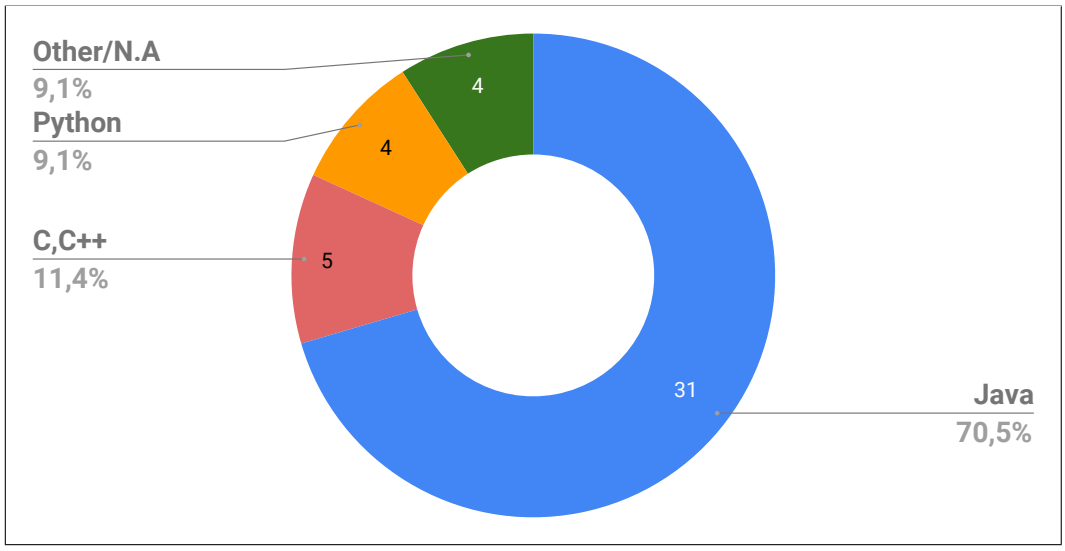


Figure 2.3: *The ratio of the programming languages used to implement simulators*

detailed information on the properties of the tools, we summarise the most valuable findings of these comparisons.

For cloud solutions, Table 2.3 shows that most of the cloud simulators apply the generic, event-driven model to simulate entities of cloud systems. We can also see that 12 out of 18 simulators are based on CloudSim. Table 2.4 depicts that only limited (mostly static) cost calculations can be performed on the majority of these tools, and mostly the bandwidth parameter is the only way to model network communications. VM management functions are less supported in the network simulator category, and they are also surprisingly simple for the cloud-specific ones, except for DISSECT-CF (offering 12 functions). The energy model is also quite simple for most solutions (using static, physical CPU power consumption values), GreenCloud and DISSECT-CF have additional possibilities. Table 2.5 reflects software quality. Java is the dominating programming language, and the source code length varies between 5 to 30 thousand in general. iCanCloud is exceptionally long, which is due to the underlying OMNET++ framework size. Code duplication and the number of files seem to be proportional to the code length; the number of bugs and vulnerabilities are the highest in the CloudSim family. The percentage of comments is surprisingly low in most cases, only TeachCloud and DISSECT-CF are close to 50%. Modelling Cloud Computing solutions is the earliest field, the CloudSim tool and its family dominate investigations in this area. Though new cloud management algorithms are still being developed and validated with them, they are rarely updated or maintained.

For solutions with IoT support, Table 2.6 shows that IoT system modelling is still not mature enough compared to Cloud Computing simulation solutions. Simulators in this group appeared around 2014, showing a wider variety of approaches (fully

simulated, semi-simulated, and real environments), and there seems to be no converging among them. Table 2.7 depicts that IOTSim and DISSECT-CF-IoT have the most detailed models reflecting almost all taxonomy elements. Important to note that geolocation support only appears in CrowdSenSim, meanwhile only DISSECT-CF-IoT supports dynamically measured IoT-side cost calculation besides cloud side-costs. The third comparison table in this group, Table 2.8 details the measured source code metrics of IoT solutions. We can see that only DISSECT-CF-IoT as an extension maintains the quality of its base simulator (having similar good values for comments and code duplication), while other extensions often worsen quality compared to their core solutions. Finally, we can see a huge variation in the lines of code metric, CrowdSenSim and DPWSim have tens of thousands of lines, while SmartSim has less than a thousand, and DISSECT-CF-IoT and MobIoTSim stand in the middle. The IoT field brought up new simulation methods: they partly affected the former cloud simulators by triggering extensions to model cloud support functionalities for IoT data management, and novel simulators also appeared to focus on the device and sensor handling capabilities of IoT systems. These solutions are not really converging, and the non-standard approaches in this area make it hard to come up with comprehensive solutions.

Considering fog modelling, according to Table 2.9 we can state that models of Fog Computing are evolving more rapidly than the ones for IoT based on the latest publications, which is approved by the fact that we found twice as many solutions for the fog group. There are many independently developed tools (half of the considered studies), and six works extend some CloudSim solution in a direct or indirect way. It also seems that former (purely) network simulation solutions are not considered anymore to be the base of new extensions for Fog Computing. Table 2.10 shows that all defined taxonomy categories are covered by at least one simulator, but the most wanting category is VM management, hence only seven out of 18 tools are able to consider the utilisation of network operations of a virtual machine. We can see a great improvement over IoT simulators for geolocation support and network models. Only the iFogSim simulator (and its variants) and the DISSECT-CF-Fog satisfy all of our categories. Unfortunately, many simulators (FogTorchΠ, OPNET or SpanEdge) aim to model one specific capability of Fog Computing, which makes them less productive and usable for general simulations. Table 2.11 depicts that the iFogSim simulator has relatively bad software quality: about one-fourth of its code is duplicated, and it has more than 25 thousand lines of code having more than 1,500 code smells, which is the worst out of all simulators. On the contrary, DISSECT-CF-Fog and PureEdgeSim have the best ratio of duplication, and EmuFog has the best ratio of comment lines. As it revealed, software re-engineering methods are highly encouraged to be used in the future to arrive at reliable and maintainable extensions.

Fog Computing modelling is the latest direction, and it tries to build on previ-

ous cloud and IoT system simulators. This area is under active research, and fog modelling is still in its infancy, mainly due to the still-forming real-world fog applications. Focusing on the currently available fog simulators, we can summarise that though usable solutions can be found for analysing specific fog capabilities or use cases, in general, complex fog environments are still hard to be addressed with a single tool. It is also unlikely that near future solutions would target coming up with a general simulator, rather extensions will appear to cover more fog management-related properties. One direction, which has already been started, is the modelling of container-based fog node behaviour, another one is the location-aware management of fog nodes. Cost modelling and energy-aware management are also missing features in many simulators, hence they still need extensive research. The sensor models are quite simple in most tools, therefore, sensor and device behaviour analysis and modelling should also be a target feature of future research.

2.4 Further Investigation of iFogSim and DISSECT-CF-Fog

Based on our detailed survey and taxonomy presented in 2.3, we can agree that all of these simulators would be interesting for further analysis. The CloudSim-based extensions (e.g. iFogSim or EdgeCloudSim) are often used for investigating Cloud and Fog Computing approaches, and in general, they are the most referred works in the literature. On the other hand, the DISSECT-CF simulator is proven to be much faster, more scalable, and more reliable than CloudSim (see [41]). This former research showed that the simulation time of DISSECT-CF is 2,800 times faster than the CloudSim simulator for similar cloud use cases. Taking into account the literature search results, the existing performance comparison of the core simulators, and the maturity and number of citations, our next goal was to make a comprehensive comparison with the original version of iFogSim and the DISSECT-CF-Fog simulator. The fog modelling capabilities of DISSECT-CF-Fog is presented comprehensively in Chapter 4.

iFogSim and DISSECT-CF-Fog are quite evolved and complex simulators, however, they follow a slightly different logic to model Fog Computing. This means that though they have similar components, we cannot match them easily. iFogSim was created to model resource management techniques in Fog environments, for which the DISSECT-CF-Fog can also be applicable. To facilitate their comparison, we gathered and compared their main properties and components and showed them in Table 2.12. Its first column names a generic simulation property or entity, the second column shows how they are represented in DISSECT-CF-Fog, and the third summarises their representation in iFogSim. As we can see, the biggest difference between them

is the chosen unit for simulation time measurement. iFogSim measures time passing in the simulated environment in milliseconds, while DISSECT-CF-Fog has a specific naming for the smallest unit for simulation time called a tick, which is related to the simulation events. The researcher using the simulator can set up the parameters and properties of a concrete simulation to associate a certain time interval (e.g. millisecond) for a tick. The measurement of processing power in the simulators can also be done with different approaches. iFogSim associates MIPS for every node, which represents the computational power and does not take into account the number of CPU cores. The number of CPU cores affects only the creation of virtual machines. In DISSECT-CF-Fog, both physical machines (PM) and virtual machines (VM) have to be configured with CPU core processing values, which define how many instructions should be processed during one tick.

Table 2.12: *Comparison of DISSECT-CF-Fog and iFogSim*

Property	DISSECT-CF-Fog	iFogSim
Unit of the simulation time	Tick	Millisecond
Unit of the processing	CPU core processing power	MIPS
Physical component	ComputingAppliance	FogNode
IoT model	Device and Sensor	Sensor
Logical component	Application	Application with AppModule and AppEdge
Task	ComputeTask on VM	Tuple
Architecture	Graph	Tree
Communication direction	Horizontal and vertical	Vertical
Dataflow	Implicit in physical connection	Separately in the AppEdge
Sensor	Processing depends on the size of the data generated	Predefined MIPS value
Pricing	Dynamic cost for cloud and IoT side	Static cost for RAM, storage, bandwidth and CPU

A physical component is represented by one dedicated class (see the third row of Table 2.12) in both simulators. To represent IoT components, iFogSim uses the Sensor class, while DISSECT-CF-Fog differentiates general IoT devices with computing and storage capacities and smaller sensors represented by the Device and Sensor classes. The logical components to define concrete applications are implemented with three classes defining processing elements and logical dataflow in iFogSim (Application, AppEdge, AppModule), which are not straightforward to configure. Besides, the ModuleMapping class is an important component, which is responsible for the mapping of the logical and physical entities based on a given strategy (e.g. cloud-aware, edge-aware). On the other hand, in DISSECT-CF-Fog, the physical topology already defines data routes, so researchers can focus on setting up the required processing units (of the components placed in the topology). The representation of computational tasks is also different. In DISSECT-CF-Fog, researchers should define a ComputeTask with a certain number of instructions, also stating the number of

instructions to be executed within a tick. In iFogSim, researchers should define a so-called Tuple for each task and state the number of MIPS required for its execution. In DISSECT-CF-Fog tasks can be dynamically created to process a certain amount of sensor-generated data, therefore, the number of instructions will be proportional (to the available data) in the created tasks. In iFogSim, a static MIPS value should be defined in the Tuple, hence it cannot respond to the actual generated data of a scenario.

Concerning the communication among components, iFogSim orders components in a hierarchical way and supports only vertical communication among elements of its layers (by default), while DISSECT-CF-Fog supports communication to any direction among any components in the topology. Figure 2.4 also depicts the different representations of IoT-Fog-Cloud systems in the considered simulators, highlighting their architectural and communication possibilities. To support cost calculations and pricing, in iFogSim static cost can be defined for CPU, bandwidth, storage, and memory usage. DISSECT-CF-Fog has a more mature cost model, and it supports XML-based configuration for cloud and IoT-side costs based on real provider pricing schemes.

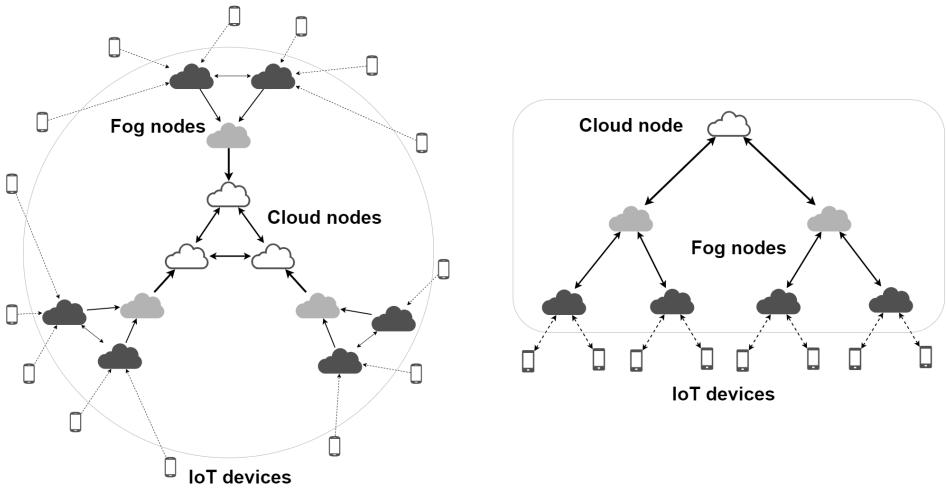


Figure 2.4: *Topology of the DISSECT-CF-Fog and the iFogSim*

2.4.1 In-depth Performance Analysis

As we mentioned in the previous section, these simulators are heterogeneous, thus we have to apply some restrictions to present a fair and realistic comparison. We limit the configuration of DISSECT-CF-Fog by allowing only-single core CPUs for the simulated resources. In the case of DISSECT-CF-Fog, the speed of the task execution depends on the number of CPU cores and their processing power, whilst in iFogSim,

only the MIPS value of the task defines the time of task processing, as we mentioned before. The common parameters that can be set up in both simulators with similar values are the following: simulation time, data generation frequency, processing power and configuration of the physical resources, count of instructions for the tasks, and finally the physical topology. Nevertheless, we cannot avoid introducing some different setups. In iFogsim, the devices have direct connections to the physical resources, while in DISSECT-CF-Fog, the actual positions and distances to the corresponding physical resources are also considered, which can affect the connections.

We also have to deal with the issue that iFogSim does not take into account the size of the generated data in task creation, because sensors in iFogSim always create Tuples with the same MIPS value, hence the file size does not have an influence on that value. As a result, dynamically received sensor data on a fog device cannot be modelled, only static, predefined tasks have to be used. To allow fair comparison, we configured the scenarios in DISSECT-CF-Fog to always generate tasks with the same size.

Concerning task forwarding, in iFogSim, a fog device uses a method to forward a received (or generated) task to a higher-level device if it cannot handle (i.e. process) it. In the case of DISSECT-CF-Fog, every application module has a threshold value to handle task overloading, which defines the number of allowed waiting tasks. If this number exceeds the threshold (so more tasks arrive than could be processed), the unhandled tasks will be forwarded to other available nodes (according to some selection algorithm). To match the default behaviour of iFogSim, the topology defined in DISSECT-CF-Fog allowed only vertical forwarding among the available fog nodes (i.e. tasks are forwarded to upper nodes only).

After applying these restrictions to make the two simulators comparable, we had to find an IoT application as a case study for our measurements. Since we thoroughly analysed meteorological applications in our research (see Chapter 3), we decided to use this analogy for the current evaluation as well. So in our scenario, sensors attached to IoT devices (i.e. weather stations) monitor weather conditions, and send the sensed data to fog or cloud resources for processing (i.e. for weather forecasting and analysis).

To perform the comparison, we defined four layers for the topology: (i) a cloud layer, (ii) an upper Fog device layer with stronger resources, (iii) a lower Fog device layer with weaker resources, and (iv) an IoT (smart) device layer. For the concrete resource parameters we defined one scenario with three different test cases:

- In the first test case, we set up 20 IoT devices to generate data to be processed;
- In the second test case we initiated 40 IoT devices;
- While in the third test case, we initiated 60 IoT devices for data generation (where each device had a single sensor).

- Concerning data processing, we used the following resource parameters for the test cases: one Cloud with 45 CPU cores and 45 GB RAM, 4 (stronger) fog nodes with 3 CPU cores and 3 GB RAM each, 20 (weaker) fog nodes with 1 CPU core and 1 GB RAM.

We did not use preset workloads during the experiments, only the started sensors generated data independently, thus in both simulators, we modelled the execution of so-called bag-of-tasks applications in fogs and clouds. In some cases the use of traditional hypervisors is not possible on fog nodes, there we may use container technologies. In our paper, we refrain from distinguishing containers and traditional virtual machines, hence both considered simulators model virtual machines to serve application execution. To be as close to iFogSim as possible, we only used one type of virtual machine in DISSECT-CF-Fog, having 1 CPU core and 1 GB RAM. In the case of iFogSim, the power of virtual machines was 1,000 MIPS. The tasks to be executed in VMs were statically set to 2,500 MIPS in both simulators. The simulation time was set to 10,000 seconds, and sensor readings were done every 5.1 seconds (i.e. the data generation frequency of the sensors). Each sensor generated 500 bytes of data during one iteration. The latency and bandwidth values were set equally in both simulators.

All the experiments were run on a PC with Intel Core i5-7300HQ 2.5GHz, 8GB RAM and a 64-bit Windows 10 operating system. The results of executing the test cases with both simulators can be seen in Table 2.13. We executed the same test cases five times with both simulators and counted their medium values to be stored in the table. To compare the use of the simulators, we only took into account the default outputs of the simulators and their execution time (e.g. cost calculations were neglected, hence they follow different logics in the simulators, and also are not really relevant for the performance comparisons).

According to these measurements, we can observe that the time needed for executing the simulation of the first test case was about ten times more with iFogSim, than with DISSECT-CF-Fog. In the second test, case we doubled the number of IoT devices, and the runtime values increased by about 25% in the case of DISSECT-CF-Fog and about 71% in the case of iFogSim. Comparing their runtime, DISSECT-CF-Fog is better suited for high-scale simulations, while iFogSim simulations become intolerably time-consuming by modelling higher than a certain number of entities. In the third test case, we could not even wait for the measurements to finish (cancelled them after 1.5 hours).

The application delay is the time within the simulation needed to process all remaining data in the system after we stopped data generation by the IoT devices. The results in Table 2.13 show that this delay was longer in the case of iFogSim, though the generated data sizes were equal for the same test cases in both simulators (hence the output results concerning the processed data were also equal). This is due

to the different methods of task creation, scheduling, and processing in the simulators (we could not eliminate all differences with the restrictions).

Finally, we used a simple source code metric to compare the implemented scenarios in the simulators. The so-called lines of code (LOC) is a common metric for analysing software quality. It is interesting to see that the same scenario could have been written three times shorter in the case of DISSECT-CF-Fog, than in iFogSim. Of course, we tried to implement the code in both simulators with the least number of methods and constructs (in Java language). We also have to state that some configuration parameters had to be set at different parts of the software (this adds some lines in the case of iFogSim, and around 20 lines of XML generation and configuration in the case of DISSECT-CF-Fog).

We can draw some conclusions from the experiments performed so far. We managed to model an IoT-Fog-Cloud environment with both simulators and investigated a meteorological IoT application execution on top of it with different sensor and fog and cloud resource numbers. While DISSECT-CF-Fog dealt with these simulations with ease, iFogSim struggled to simulate more than 65 entities of this complex system. Nevertheless, it is obvious that there are only a small number of real-world IoT applications that require only hundreds of sensors and fog or cloud resources; we need to be able to examine systems and applications composed of hundreds of thousands of these components. [P11] presents the related large-scale evaluation in more detail.

Table 2.13: *Comparison of the two simulators*

Property	DISSECT-CF-Fog			iFogSim		
Test case	I.	II.	III.	I.	II.	III.
Runtime (ms)	248.75	312.5	392.58	2,260.33	3,873.66	5,400,000*
Application delay (min)	3.41	4.33	4.33	14.89	17.52	N.A.
Generated data (byte)	19,600,000	39,200,000	58,800,000	19,600,000	39,200,000	N.A.
Lines of code	50 lines + XML files for detailed configuration			159 lines + some inline constants		

Since the result of the first scenario showed a strong performance difference between the investigated simulators, we decided to perform further examinations. Next, we applied JProfiler[6], which is able to analyse Java-based applications considering threads, classes, instances, and usage of the garbage collector, besides memory and CPU usage. We repeated the test cases defined for both simulators in the previous section focusing on these metrics and characteristics as shown in Figure 2.5. Each row presents the three test cases of the first scenario in the corresponding simulator. The comparison is based on the following characteristics: Memory reflects the heap memory, the used size is labelled by blue, whilst the free size of memory

[6]JProfiler (accessed in June, 2020): https://www.ej-technologies.com/products/jprofiler/overview.html

is labelled by green. Recorded Objects present the instantiated objects, blue refers to the number of arrays, and green refers to the non-arrays objects. The Recorded Throughput shows the freed objects per second using green colour, and blue represents created objects per second. The GC Activity presents the activity ratio of the garbage collector of the JVM, Thread presents the number of threads with runnable state, whilst Classes shows the number of used classes during the evaluation. Finally, CPU Load reflects the process load (green) and the system load (blue).

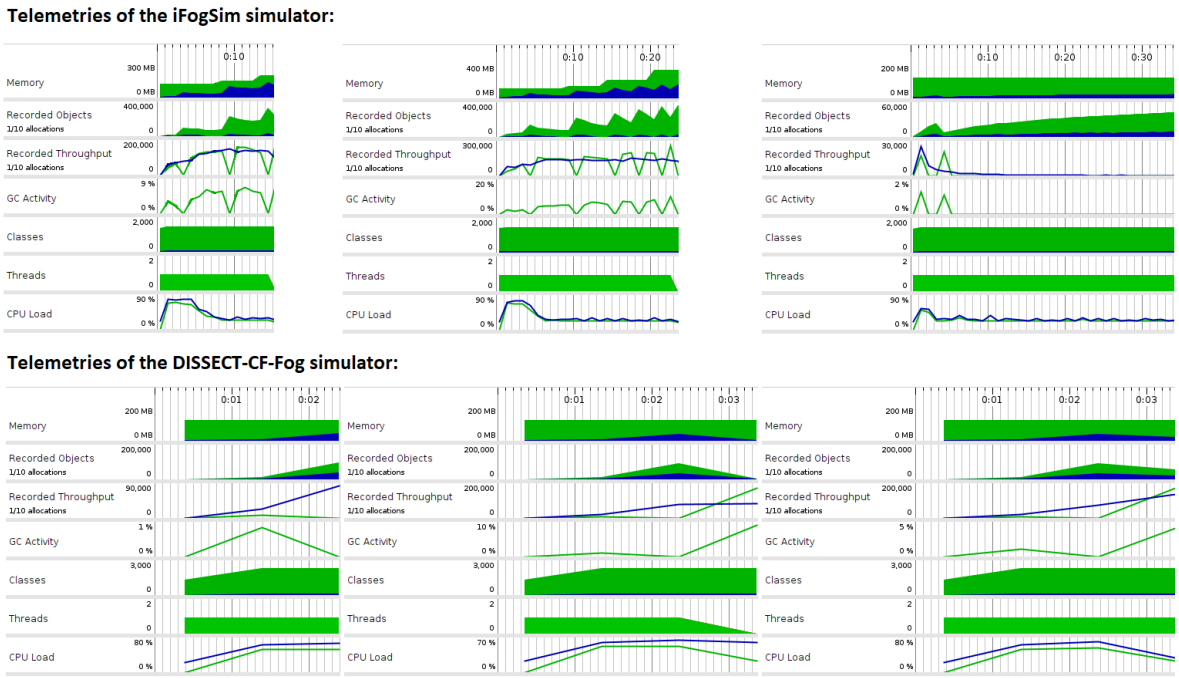


Figure 2.5: *Telemetry data of the investigated simulators*

Interpreting the results, DISSECT-CF-Fog utilises less memory, and in all test cases, the heap size stayed less than 200 MBs, whilst in the case of iFogSim the heap size of the second test case almost reached 400 MBs. The Recorded Object value was almost four-times higher during the evaluation with iFogSim, however, DISSECT-CF-Fog used almost 3,000 Java classes for the evaluation (external libraries are considered by the JProfiler, as well). The iFogSim tool used the CPU more intensively than DISSECT-CF-Fog; the CPU Load almost reached 90% in the first two test cases during the iFogSim simulations. The GC Activity and the Recorded Throughput metrics point out a possible malfunction in the third test case of iFogSim, because after about 5 seconds, these values were correlated showing no relevant operation occurrence. The reason for this behaviour is the process starvation caused by the Java Finalizer thread. A similar issue was mentioned in [2], however, this problem can strongly relate to the negative code quality as it has been presented in Table 2.11.

2.5 Discussion and Concluding Remarks

In the past decade, we have seen how the latest technological advances shaped distributed systems and led to the emergence of clouds, IoT, and finally fog systems. The complex networks and environments they established are closely coupled: the data generated by IoT devices have to be stored, processed, and analysed by cloud or fog services to ensure reliability and sustainability. We can find numerous simulators for these purposes, but in many cases, it is hard to reveal their differences, and implementing our use cases with different solutions is time-consuming.

By responding to these challenges, we proposed a survey and taxonomy of the available simulators modelling clouds, IoT, and fogs. The novelty of our work lies in the applied viewpoints to perform the classifications. In our taxonomy, we separated the considered simulators into three groups, and presented comparison tables based on the taxonomy to reveal their differences and to highlight how they model the elements of IoT-Fog-Cloud systems. We can conclude that a comprehensive model for Fog Computing is still wanting, and complex fog environments are hard to be addressed with a single simulator. Our main recommendation for further research is to continue model extensions of simulators to better grasp fog capabilities with the warning to maintain software quality. This way can lead to easier understanding and shorter learning curves for designing and performing experimentation in the field.

We also compared two fog modelling approaches, namely iFogSim and DISSECT-CF-Fog and presented an evaluation, which showed how to create and execute simulated IoT scenarios using fog and cloud resources with these tools. Our results highlight that DISSECT-CF-Fog can provide easier configurations and faster and more reliable simulations for higher scales.

The results of this chapter belong to **Thesis I**, and its contents were published in papers [P3], [P7], and [P11]. My contributions presented in this chapter are the following:

I/1. I proposed a comprehensive survey and taxonomy of numerous simulation approaches aiming at the examination of cloud, IoT, and fog systems.

I/2. I compared and analysed these simulation environments in terms of functionality, usability, maintainability, and code quality in order to determine the most relevant properties for modelling IoT-Fog-Cloud systems.

I/3. I presented an in-depth performance analysis with a comparison of the two most prominent simulators in these fields, namely DISSECT-CF-Fog and iFogSim, showing a significant performance difference in favour of DISSECT-CF-Fog.

3

Simulating IoT Systems in a Multi-cloud Environment

3.1 Introduction

The Internet of Things (IoT) paradigm allows for interconnecting sensors (e.g. heart rate, heat, motion, etc.) and actuators (e.g. motors or lighting devices) in automated and customisable systems [62]. IoT systems are currently expanding rapidly as the amount of smart devices (sensors with networking capabilities) is growing substantially, and at the same time, the costs of sensors are decreasing.

IoT solutions are often used a lot within businesses to increase the performance in certain areas, and allow for smarter decisions to be made based on more accurate and valuable data. Businesses have grown to require IoT systems to be accurate, as decisions based on their data are heavily relied on. However, many sensors have different behavioural patterns. For example, a heart rate sensor has different behaviour compared to a light sensor, since a heart rate sensor relies on human behaviour as well, which is inherently unpredictable, whereas a light sensor could be predicted quite accurately based on the time of day and location. Predicting how a sensor may impact a system is important as companies generally want to leverage the most out of an IoT system. An incorrect estimation of their performance could possibly have a negative impact on the performance of other systems (e.g. using too many sensors could flood the network, potentially causing inaccurate data, slow response times, or even system crashes). As there are many ways a sensor can behave, it is

difficult to predict the effects they may have on the overall system, therefore, their interaction and operation must be analysed. Performing such testing could be costly, time-consuming, and high-risk if the infrastructure has to be created and a wide range of sensors are purchased before any information is obtained about the planned system behaviour. It is even more difficult to determine the impact of a prototype system on the network, as there may be limited or no physical sensors available to perform the tests with. An example of this issue is the introduction of soil moisture sensors that analyse soil properties in real-time and adjust water sprinklers to ensure crops have the correct conditions to grow. In order to test this IoT system efficiently, a huge number of sensors is required, however, they can become quite costly and difficult to implement.

As it is discussed in Section 2.3, there are many simulators available to examine distributed, and specifically cloud systems, as well as IoT environments. In this chapter, we lay the foundations for the flexible and scalable modelling of IoT sensors through our extensions made to the DISSECT-CF simulator.

The remainder of this chapter is as follows: Section 3.2 first presents the description of the IoT extension, then in its subsections, the pricing model is described along with cost analysis scenarios. In Section 3.3, the multi-cloud extension is presented with detailed emphasis on the possible device allocation strategies. Finally, Section 3.4 concludes the chapter and highlights the main contributions.

3.2 The Proposed Pricing-aware Model for IoT Sensors and Applications

We aim at supporting the simulation of up to thousands of devices participating in previously unforeseen/existing IoT scenarios that have not been examined before in detail (e.g. in terms of scalability, energy efficiency or management costs). Sensors are essential parts of IoT systems, and they are usually passive entities. Their performance is limited by their network gateway's (i.e. the device which polls for the measurements and sends them away) connectivity and maximum update frequency. Actuators are entities also limited by their network connectivity and reaction time (e.g. how long does it take to actually perform an actuation action). They also have a unique feature that allows changing the location of non-cloud entities. Finally, central computing services provide the large-scale background processing and storage capabilities needed for IoT scenarios. According to recent advances in IoT, these services are expected to be used only if unavoidable. The simulator extension presented in this chapter takes into account the following IoT components: sensors, devices (i.e. gateways or brokers), and applications (deployed in a central computing service, i.e. cloud).

To represent an IoT device with sensors, the following parameters are considered: it has a unique identifier and its lifetime is specified with start time and stop time. The cardinality of the supervised sensor set must be defined as well. Alongside the set cardinality, the average data produced by one of the sensors can be specified in the set. Data generation frequency could be set for the sensors (e.g. in milliseconds) to determine the time interval between two measurements. Each device controls its local storage with a predefined capacity and its network connectivity to the outside world is specified by bandwidth and latency values.

IoT applications that receive, process, and store IoT data can also be modelled by installing on the IaaS cloud representation layer of DISSECT-CF. The following properties are taken into consideration by modelling an IoT application: the types of virtual machines utilised by the application and the task size attribute, which aggregates IoT data into a processing unit for the VM.

3.2.1 IoT Pricing Schemes

In our simulation model, we aim to investigate certain IoT cloud applications, therefore, we need to define and monitor the following parameters: the number of sensors or devices used, the total number of messages (and their data) sent in a certain period of time, and the uptime and capacity of virtual machines used to provide ingest services. Based on these parameters we can estimate how our application would be charged after operating its system for a certain amount of time at a concrete IoT cloud provider.

The calculation of the prices depends on different methods. Some providers bill only according to the number of messages sent, while others also charge for the number of devices used. The situation is very similar if we consider the virtual machine rental or application service prices. One can be charged after GB-hour (uptime) or according to a fixed monthly service price. This price also depends on the configuration of the virtual machine or the selected application service, especially the amount of RAM used or the number of CPU cores or their clock signal.

We considered the following most popular providers as the base of our extension: (i) Microsoft and its IoT platform called Azure IoT Hub[7], (ii) IBM's Bluemix IoT platform[8], the services of (iii) Amazon (AWS IoT)[9], and (iv) Oracle's IoT platform[10]. We took into account the prices publicly available on the websites of the providers and if it necessary we asked for clarifications via email.

[7]MS Azure IoT Hub (accessed in January, 2017): https://azure.microsoft.com/en-us/services/iot-hub

[8]IBM Bluemix (Accessed at January, 2017): https://www.ibm.com/cloud-computing/bluemix/internet-of-things

[9]Amazon AWS IoT (accessed in January, 2017): https://aws.amazon.com/iot/pricing

[10]Oracle IoT platform (accessed in January, 2017): https://cloud.oracle.com/en_US/opc/iot/pricing

Azure IoT Hub

Concerning the IoT-side pricing, Azure IoT Hub charges after the chosen edition/tier. This means that there are intervals for the number of messages used in a month. Azure also comes with some additional platform services similar to other providers, but we only kept the general parts to allow the identification of common pricing components. There is a restriction for message sizes, which depends on the chosen tier. One can choose from four tiers, Free, S1, S2, and S3. Each of them varies in price and the total messages allowed per day. The message and group size of the Free tier are significantly more limited compared to the other tiers.

IBM Bluemix

IBM Bluemix IoT platform's pricing follows the pay-as-you-go approach. Bluemix only charges after the MB of data exchanged. They differentiate three categories in terms of data usage and each of them comes with a different price per MB. The more we transfer the less our price per MB will be.

Amazon's IoT Platform

Amazon's IoT platform can also be classified as a pay-as-you-go service. Prices have two components: publishing cost (the number of messages published to AWS IoT) and delivery cost (the number of messages delivered by AWS IoT to devices or applications). A message is a 512-byte block of data and the pricing in EU and US regions denotes 5 US dollars per million messages. In addition, there is no charge for deliveries to some other AWS Services.

Oracle's IoT Platform

Finally, we investigated the pricing of Oracle's IoT solution. This pricing method is slightly different from the three providers described before. It is more similar to Azure's tiers than to the completely "pay as you go" billing like in Bluemix. The information was gathered from and calculated with the so-called Metered Services. There are four product categories regarding the used devices (wearable, consumer, telematics, and business). Categories determine the monthly device price and the number of messages that can be sent by that particular type of device. In addition, there is a restriction on how many messages a particular type of device delivers per month. In case the number of messages sent by a device is more than the device's category permits, an additional price will be charged according to a predefined price per thousand messages.

3.2.2 Cloud Pricing Schemes

Besides the IoT-side costs, we had to investigate how the four providers calculate the cloud-side costs. Most providers have a simple method, which is the following: (i) to run an IoT application one needs at least one virtual machine (VM), container, compute service, or application instance, which has a fixed instance price every month, or (ii) the providers consider the hour per price for every instance the IoT application needs. The pricing scheme of these providers can be found on their websites. We considered Azure's application service[11], Bluemix's runtime pricing sheet under the Runtimes section[12], Amazon EC2 On-Demand prices[13], and Oracle's compute service[14] together with the Metered Services pricing calculator[15]. The cloud cost is based on either instance prices (Azure and Oracle), hourly prices (Amazon), or the mix of the two (Bluemix), in which the provider uses both types of price calculating. For example, Oracle charges depending on the daily uptime of our application as well as the number of CPU cores used by our VMs.

3.2.3 The Weather Forecasting IoT Use Case

In Figure 3.1, we reveal the typical data flow of a weather forecasting service, which we often use to evaluate the different versions of our simulator extension. This application aims to make weather analysis more efficient by allowing the purchase of a small weather station kit including light sensors (to potentially capture cloud coverage), wind sensors (to collect wind speed), and temperature sensors (to capture the current ambient temperature). The weather station will then create a summary of the sensor's findings over a certain period of time and report it to a cloud service for further processing, such as detecting hurricanes or heat waves in the early stages. If many of these stations are set up over a region, it can provide accurate and detailed data flow to the cloud service to produce accurate results.

As one of the earliest examples of sensor networks is from the field of meteorology and weather prediction, we chose to model the crowd-sourced meteorological service of Hungary called Idokep.hu[16]. It was established in 2004, and it is one of the most popular meteorology websites in Hungary. Detailed information on its system architecture and operation can also be found on its website: more than 400 stations send

[11]MS Azure price calculator (accessed in January, 2017): https://azure.microsoft.com/en-gb/pricing/calculator/

[12]IBM Bluemix pricing sheet (accessed in January, 2017): https://www.ibm.com/cloud-computing/bluemix

[13]Amazon pricing (accessed in January, 2017): https://aws.amazon.com/ec2/pricing/on-demand/

[14]Oracle pricing (accessed in January, 2017): https://cloud.oracle.com/en_US/opc/compute/compute/pricing

[15]Orcale Metered Services pricing calculator (accessed in January, 2017): https://shop.oracle.com/cloudstore/index.html?product=compute

[16]Idokep.hu (accessed in February, 2017): http://www.idokep.hu

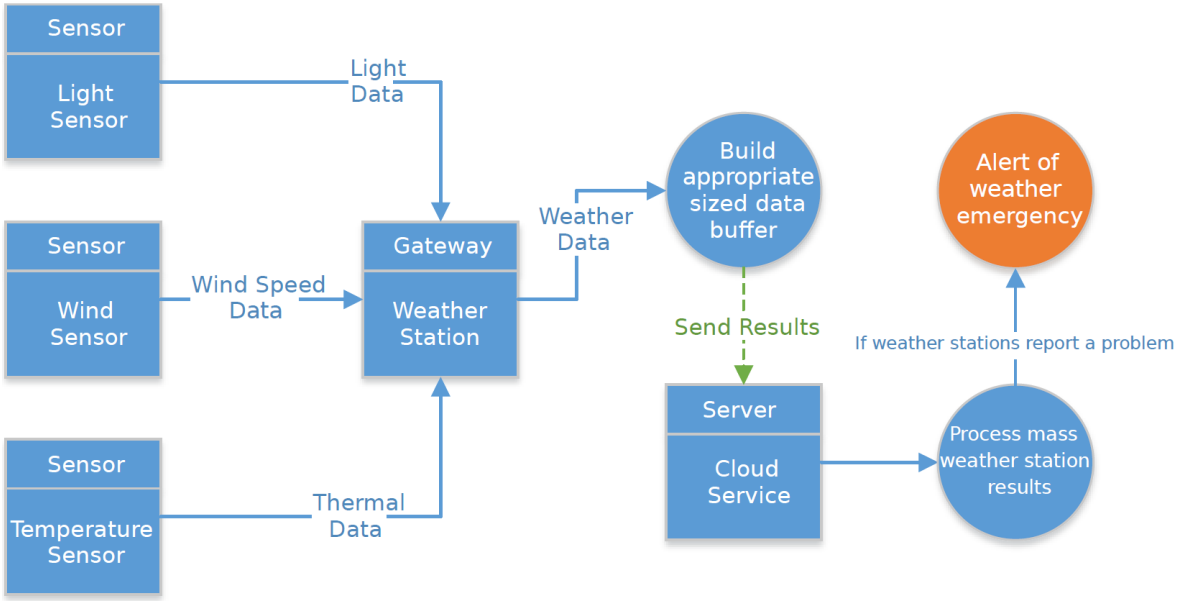


Figure 3.1: *A typical IoT use case: meteorological application*

sensor data to their system (including temperature, humidity, barometric pressure, rainfall, and wind properties), and the actual weather conditions are refreshed every 10 minutes. They also provide forecasts for up to a week. They also produce and sell sensor stations capable of extending their sensor network and improving their weather predictions to be installed at buyer-specific locations. Detailed information on the properties of this IoT Cloud application operated over a month can be seen in Table 3.1.

Table 3.1: *Basic configuration information of the application*

Devices with sensors	3,864
Device type	Consumer
Message size (KB)	0.05
Messages / month / device	4,464
Total messages / day	556,416
Total messages / month	17,248,896
MB exchanged / month	842.23125
Messages transferred / device / hour	6
Test duration (days)	31
Full uptime (hours)	744

Based on the investigations presented in the previous section, we performed a cost estimation of operating this meteorological application for a period of one month. Table 3.2 shows the prices to be paid by using these providers. In these calculations,

we considered 483 stations with 3864 sensors. From this figure, we can see that Azure charges ∼610 Euros. We estimated this cost concerning the IoT-related prices (for sending messages) and the price of running the application service (gathering and processing the messages in the system). Bluemix offers a reasonably better price. The IoT part charges after MBs are exchanged, in our case ∼0.82 Euros, while the required computational services cost ∼284 Euros. Amazon offers the best price at ∼241 Euros, while Oracle is incredibly expensive compared to the other providers at ∼3862 Euros according to our estimations. The number is so high because of the high number of devices used. The device price is ∼3594 Euros which is the greater part of the total price. We do not need to pay for additional messages since none of the devices exceeds the restriction for message numbers of consumer product type.

Table 3.2: *Cost estimation for the meteorological case study*

Provider / Cost	Azure	Cost	Bluemix	Cost	Amazon	Cost	Oracle	Cost
IoT fix prices and device side								
Device price / month	-		-		-		+	3,593.52
”Price / message” pricing	-		-		+	78.69	+	
”X messages / month” pricing	+	421.65	-		-		+	0
Data exchange (in MB)	-		+	0.82	-		-	
Message size limit	4		-		0.5		-	
Total messages / day with size limit	556,416				552,960			
Cloud side								
Instance prices	+	188.22	+	245.38	-		+	268
GB-hour prices	-		0.05	39.13	0.01	162.53	-	
TOTAL PRICE / MONTH		609.87		285.33		241.22		3,861.52

3.2.4 Evaluation with the Pricing-aware IoT Extension

The weather forecasting scenarios play a vital role in the evaluation phase of our publications, therefore, we used it as a base point to execute various IoT scenarios. The execution steps are listed below:

1. Set up the IaaS clouds. In our evaluations, we typically used the model of a Hungarian private infrastructure, namely the ELKH Cloud [28] (formerly the LPDS Cloud of MTA SZTAKI).

2. Set up the necessary amount of IoT devices and sensors.

3. Initialise VM parameters and IoT applications. On each IoT application, an initial VM is deployed for data processing and the metering process in all IoT devices is also started.

4. The IoT devices then monitor their environment, they then save and send sensor data (to the cloud storage). Parallel to this, methods responsible for calculating cost estimate the price of IoT and cloud operation, based on the generated data and processing of those data. The process of cost calculation depends on the chosen provider. If provider pricing is not time-dependent, like in the case of Bluemix, we have to pay only after data traffic, then this loop is executed only once, at the end of the simulation. Otherwise, if the provider cost is time-dependent (e.g. having daily or monthly fees), the call of the related methods is recurring based on the time interval.

5. A daemon service checks regularly if the cloud repository received a scenario-specific amount of data. If so, then the IoT application generates the computer tasks.

6. Next, for each generated task, a free VM is searched. If a VM is found, the task and the relevant data are sent to it for processing.

7. In case no free VMs are found, the daemon initiates a new VM deployment and holds back the not yet mapped tasks.

8. If at the end of the task assignment phase, there are still free VMs, they are all decommissioned except those that are held back for the next rounds (this amount can be configured and even completely turned off at will). Step 7-8 implements the auto-scaler for the application.

9. Finally, the execution returns to Step 5 until there is unprocessed data.

In the next two scenarios, we mainly focus on how resource utilisation and management patterns alter based on changing sensor behaviour, and how these affect the incurred costs of operating the IoT system (e.g. how different sensor data sizes and the varying number of stations and sensors affect the operation of the simulated IoT system). The aim of these scenarios was to focus on the validation of our proposed IoT extensions, and thus the used scenarios abstract away some low-level properties of a real-world weather forecasting service.

Before getting into the details, we clarify the common behavioural patterns we used during all of the scenarios below (these were the common starting points for all scenarios unless stated otherwise). First of all, to limit simulation runtime, all of our experiments limited the station lifetimes to a single day. A station typically utilised 8 sensors and a sensor measurement was executed every minute. The start-up period of the stations was selected randomly between 0 and 20 minutes. The task creator daemon service spawned tasks after the cloud storage received the metering data, but the size of a task could not exceed 250 KB. This step ensured the estimated processing time of 5 minutes/task. If a task was started with less than 250 KB to

process, the execution time was scaled down. Finally, the application was using Bluemix VM instances utilising 8 CPU cores and 4 GB memory and its cost was set based on Table 3.2.

First Scenario

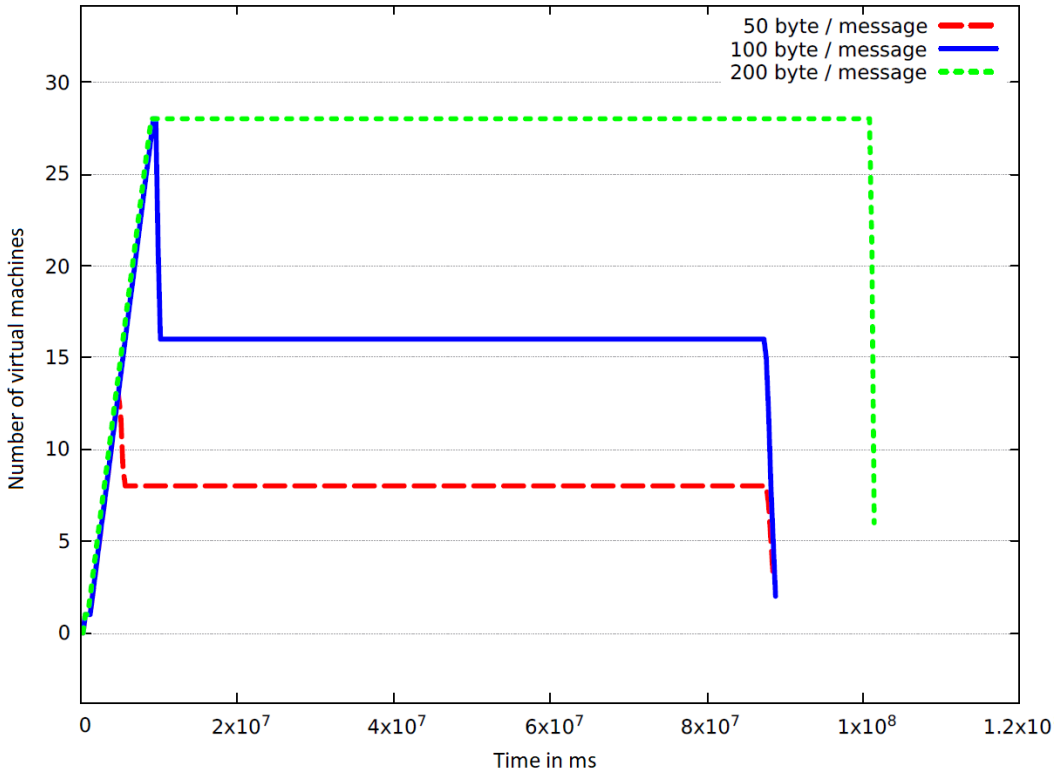


Figure 3.2: *Number of virtual machines in the first scenario*

Table 3.3: *Number of VMs, tasks, and the amount produced data in the first scenario*

Amount of data (byte)	Number of VMs	Number of tasks	Produced data (GB)
50	12	1,153	0.261
100	27	2,299	0.522
200	28	4,486	1.044

In the first scenario, we varied the amount of data produced by the sensors of 486 weather stations: we set 50, 100, and 200 bytes for different cases (allowing overheads for storage, network transfer, different data formats, and secure encoding etc.). We also investigated how the costs of the IoT side changed if we would use

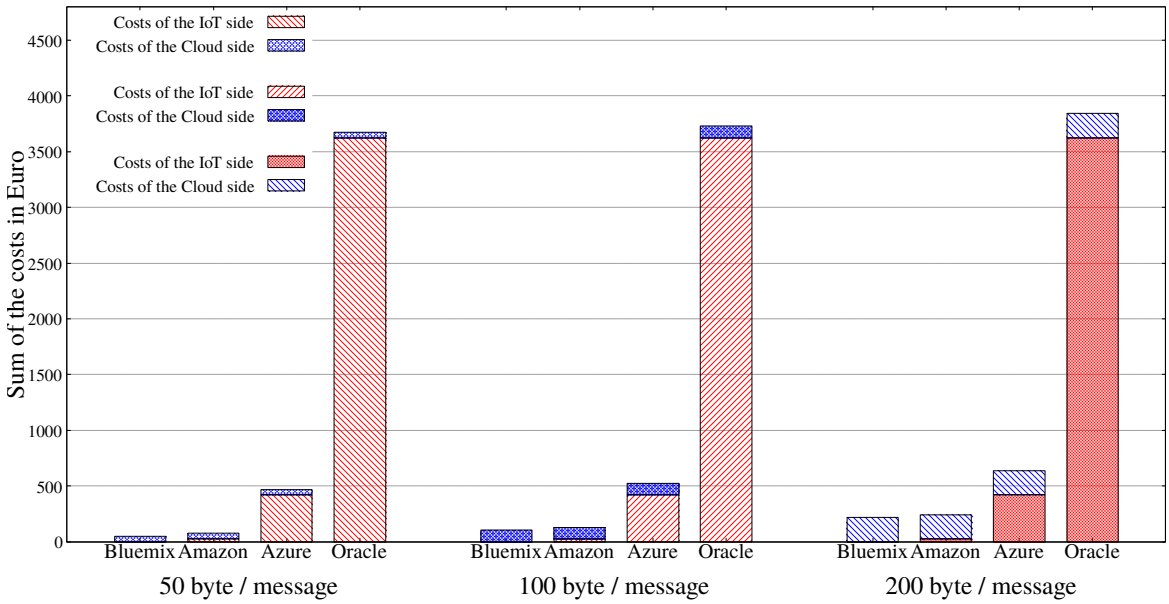


Figure 3.3: *IoT and cloud costs in the first scenario*

Table 3.4: *IoT and cloud costs in the second scenario*

IoT provider	Bluemix		Amazon		Oracle		Azure	
IoT-side cost	0.18		18.92		14,136.00		421.65	
VM function	ON	OFF	ON	OFF	ON	OFF	ON	OFF
Bluemix cloud cost	51.80	89.39	51.80	89.39	51.80	89.39	51.80	89.39
Sum (Euro)	**51.98**	89.58	70.72	108.31	14,187.80	14,225.39	473.45	511.04

one of the four IoT providers defined before. The measurement results can be seen in Figure 3.2, Figure 3.3, and Table 3.3.

For the first case with 50 bytes of sensor data, we measured 0.261 GBs of produced data in total, while in the second case of 100 bytes, we measured 0.522 GBs, and in the third of 200 bytes, we measured 1.044 GBs (showing linear scale up). In the three use cases, we needed 12, 27, and 28 VMs to process all tasks respectively. With the preloaded cloud parameters, the system is allowed to start a maximum of 28 virtual machines, therefore, in the first case of 50 bytes our cloud cost was 48.839 Euros, in the second case of 100 bytes the cloud cost was 103.896 Euros, and finally, in the last case, our cloud cost was 217.856 Euros. The lesson learnt from this scenario is that if we use more than 200 bytes per message, we need stronger virtual machines (also a larger cloud with stronger physical resources) to manage our application, because in the third case the simulation run for more than 24 hours (despite the fact that the sensors were only producing data for a single day), which increased our costs using time-dependent cloud services. Finally, Table 3.3 shows how many

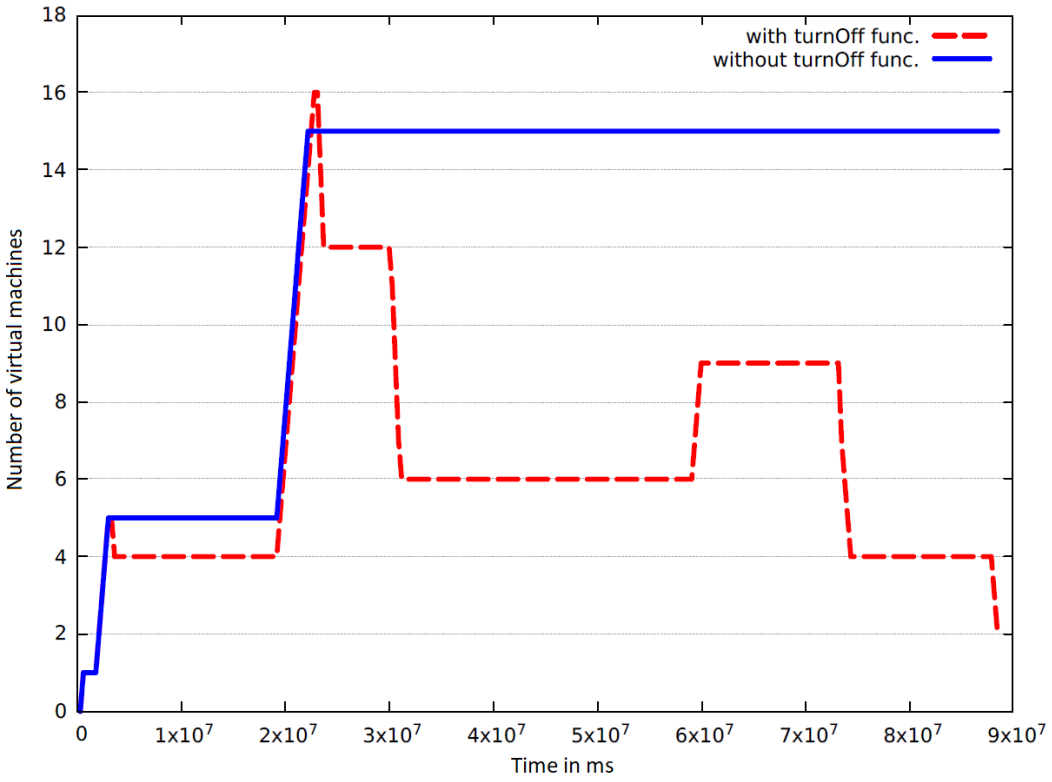


Figure 3.4: *Number of virtual machines in the first scenario*

virtual machines are needed to process all of the generated data for all test cases, and how many tasks were generated for the produced data.

Figure 3.3 presents a cost comparison for all considered providers. We can see that Oracle costs are much higher than the other three providers in all cases (50, 100, 200 bytes messages). The main cause of this issue is that Oracle charges after each utilised device, which is not the case for other providers. Our initial estimations show that only such an IoT cloud system operation is beneficial with Oracle, which has at most 200 devices and transfers 1-2 messages per minute per device.

Second Scenario

As we model a crowd-sourced service, we expect to see more dynamic behaviour regarding the number of active stations. In the previous case, we used a static number of stations per experiment, while in this scenario, we ensured that the station numbers dynamically changed. Such changes may occur due to station or sensor failures, or even sensor replacement. In this scenario, we performed these changes at specific hours of the day: from 0-5:00 a.m. we started 200 stations, from 5-8:00 a.m. we operated 700 stations, from 8:00 a.m. to 4:00 p.m. we scaled them down to 300, then from 4-8:00 p.m. up to 500, finally for the last round from 8-12:00

p.m. we set it back to 200. In this experiment we also wanted to examine the effects of VM decommissioning; therefore, we executed two different cases, one with and one without turning off unused VMs. The results can be seen in Table 3.4. We can see that without turning off the unused VMs from 6:00 p.m. we kept 15 VMs alive (resulting in more over-provisioning), while in the other case, the number of running VMs dynamically changed to the one required by the number of tasks to be processed. Figure 3.4 shows what happens with the application operating costs if we do not turn off the unused, but still running virtual machines. The cheapest IoT provider is Bluemix with 51.98 Euros, and we can save almost 38 Euros using the VM turning off function.

In summary, we have shown that with our extended DISSECT-CF-IoT simulator, we can investigate the behaviour and operating costs of these systems and contribute to the development of better design and management solutions in this research field.

3.3 A Multi-cloud Simulation Environment

The main research question of this section is how we can influence the behaviour of an IoT application if the sensors can have different allocation strategies for multiple clouds. In the earlier version of the extended DISSECT-CF-IoT we could only exploit one cloud data centre to start VMs, therefore, all sensors and smart devices were connected to this specific cloud, and all the generated data of the sensors were processed by virtual machines running in the same cloud (as we summarised in the previous section). In this single cloud setup, a cloud can have a preloaded cost calculation policy with a single pricing scheme. Smart devices usually have different sensors and usage frequencies affecting data generation methods that can influence cloud service operation and also the provider pricing. As a result, a single cloud could be easily overloaded, and the unprocessed data could hinder the operation of the IoT application causing longer response times and even service unavailability in real-world services. In this section, we introduce the possibility of multi-cloud management for IoT cloud simulations in DISSECT-CF-IoT.

During the start of the simulation, we can set up different IaaS clouds, and another improvement is the introduction of a cloud broker, which can manage different IoT applications and their VM queues. These queues may have virtual machines with different pricing policies, and within a simulation the broker can decide to which cloud (and to which application) the IoT devices should be connected, thus where the generated data should be sent and processed in an application. This revised IoT Cloud management architecture is depicted in Figure 3.5, showing one cloud with three different applications mapped to three different VM queues. These extensions make the simulator more flexible and capable of performing scalability experiments involving multiple cloud providers.

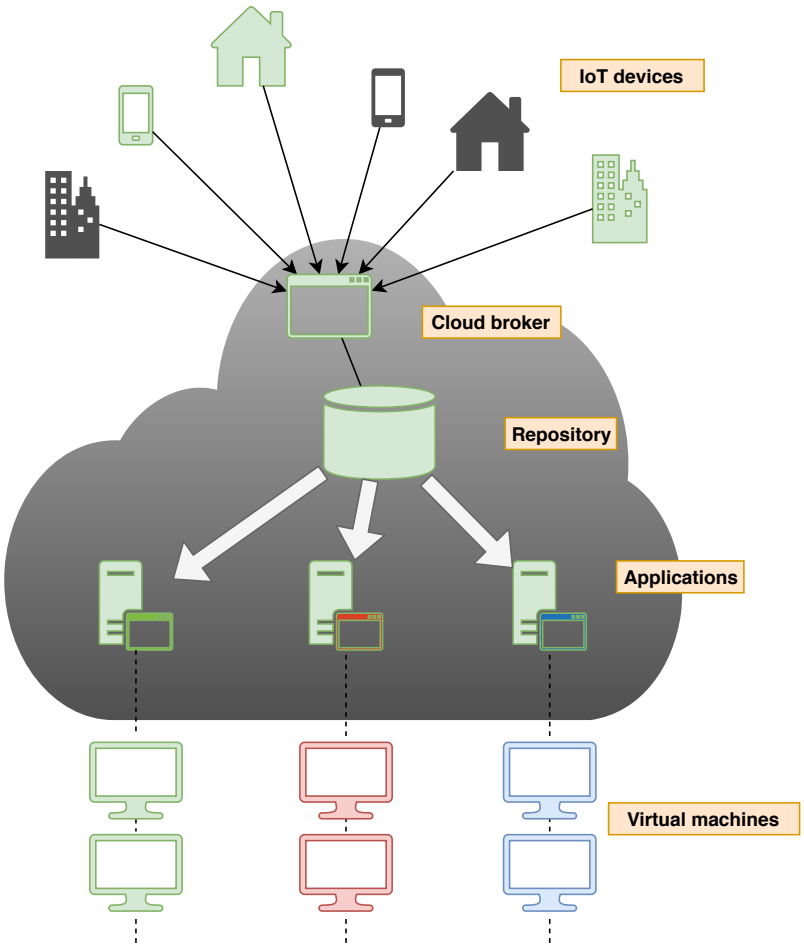


Figure 3.5: *The general IoT execution flow in the extended DISSECT-CF-IoT simulator*

In the IoT paradigm, the sensors are passive entities of the systems, thus their performance is limited by the operation frequency (i.e. data generation, storing, transfer to the cloud), up-time and network connection. Usually, large amounts of sensor data are sent from the smart devices to cloud resources for further computation and analysis. Since resource consumption can be costly, IoT application owners can reduce their expenses by selecting a provider having a suitable pricing scheme.

Currently, four different resource allocation (i.e. device) strategies can be chosen to perform cloud provider selection during each IoT device start-up; they are the following: (i) *Random*, (ii) *Cost-aware*, (iii) *Runtime-aware*, and (iv) *Pliant*.

3.3.1 Basic Strategies

With the *Random* strategy, the cloud broker chooses one of the available applications running in the simulated clouds randomly for an actual IoT device (sensor or station).

The *Cost-aware* strategy looks for the cheapest available VM in a cloud (based on

their static pricing properties), thus it compares the prices of the required VM flavors for a given device. This solution may be more suitable for IoT applications having relatively small data processing needs or that are less susceptible to the processing time because cloud providers usually offer lower resource capacities for lower costs.

In the *Load-balanced* strategy, the corresponding algorithm ranks the available applications (residing in different clouds) by a specific value defined by the ratio of the number of already connected devices and the number of available physical machines of the hosting cloud. This is a dynamic strategy that takes into account the actual load of the available clouds. Applications having longer data processing needs may prefer this strategy.

3.3.2 The Pliant Strategy

Fuzzy sets were introduced in 1965 with the aim of reconciling mathematical modelling and human knowledge in the engineering sciences. Fuzzy logic means that we cannot decide whether the value is true or not. The truth lies between the true and false values, hence it offers more flexibility for reasoning [75]. Over the last century, fuzzy sets and fuzzy logic [16] have become more popular areas for research, and they are being applied in fields such as computer science, mathematics, and engineering. This has led to truly enormous literature, where there are presently over thirty thousand published papers dealing with fuzzy logic, and several hundreds of books have appeared on the various facets of the theory and the methodology. However, there is not a single, superior fuzzy logic or fuzzy reasoning method available, although there are numerous competing theories.

The Pliant system is a kind of fuzzy theory that is similar to a fuzzy system [17]. The difference between the two systems lies in the choice of operators. In fuzzy theory, the membership function plays an important role, but the exact definition of this function is often unclear. In the Pliant system, we use a so-called distending function, which represents a soft inequality. Furthermore, the various operators, which are called conjunction, disjunction, and aggregative operators, are closely related to each other. We also have a generator function, which can be used to create such operators. In the Pliant system, the corresponding aggregative operators of the strict t-norm and strict t-conorm are equivalent, and De Morgan's law is obeyed with the corresponding strong negation of the strict t-norm or t-conorm.

The Pliant system has a strict, monotonously increasing t-norm and t-conorm, and the following expression is valid for the generator function:

$$f_c(x) f_d(x) = 1, \tag{3.1}$$

where $f_c(x)$ and $f_d(x)$ are the generator functions for the conjunctive and disjunctive logical operators, respectively. This system is defined in the [0,1] interval.

The operators of the Pliant system are

$$c(\mathbf{x}) = \frac{1}{1 + \left(\sum_{i=1}^{n} w_i \left(\frac{1-x_i}{x_i} \right)^{\alpha} \right)^{1/\alpha}} \tag{3.2}$$

$$d(\mathbf{x}) = \frac{1}{1 + \left(\sum_{i=1}^{n} w_i \left(\frac{1-x_i}{x_i} \right)^{-\alpha} \right)^{-1/\alpha}} \tag{3.3}$$

$$a_{\nu_*}(\mathbf{x}) = \frac{1}{1 + \left(\frac{1-\nu_*}{\nu_*} \right) \prod_{i=1}^{n} \left(\frac{1-x_i}{x_i} \frac{1-\nu_*}{\nu_*} \right)^{w_i}} \tag{3.4}$$

$$n(x) = \frac{1}{1 + \left(\frac{1-\nu_*}{\nu_*} \right)^2 \frac{x}{1-x}}, \tag{3.5}$$

$$\kappa_{\nu}^{(\lambda)}(x) = \frac{1}{1 + \frac{1-\nu_0}{\nu_0} \left(\frac{\nu}{1-\nu} \frac{1-x}{x} \right)^{\lambda}} \tag{3.6}$$

where $\nu_* \in]0, 1[$, with generator functions

$$f_c(x) = \left(\frac{1-x}{x} \right)^{\alpha} \qquad f_d(x) = \left(\frac{1-x}{x} \right)^{-\alpha}, \tag{3.7}$$

where $\alpha > 0$.

The operators c, d, and n fulfil the De Morgan identity for all ν, a and n fulfil the self-De Morgan identity for all ν, and the aggregative operator is distributive with the strict t-norm or t-conorm. The ν value expresses the expected value of the given context. This means that if the given x value is greater than ν, then the operators increase the value of x. The opposite is true when x is smaller than ν.

In a previous work [33], we used the Pliant system approach to schedule applications to VMs in a cloud by minimising energy consumption. There, we experienced that uncertainty could be well tolerated with this approach, and better results can be achieved with this model than with traditional approaches.

In this work, we create a new algorithm that can predict which cloud could be the best for managing a given IoT device by calculating a score for each cloud using the environment properties. This algorithm is also based on the Pliant logic, therefore, for each cloud (i.e. for each VM queue in a cloud), it calculates a score number. The first step of the algorithm is to normalise the data into the [0,1] interval. We apply a Sigmoid function for this purpose, which is one of the most popular threshold functions. $\lambda > 0$ determines the steepness of the continuous function f and squashes the content of the function:

Table 3.5: *Normalisation parameters*

Parameter	Lambda	Shift
General VM cost	-1.0/96.0	15
Cost of the application	-1.0	(Maximum price - minimum price) /2
Workload	-1.0	Maximum workload
Number of VMs	-1.0/8.0	3
Number of stations	-0.125	Number of stations / number of applications
Number of active stations	-0.125	Number of stations / active stations
VM memory size	1.0/256	350
VM CPU cores	1.0/32	3

$$f_{shift,\lambda}(x) = \frac{1}{1 + e^{-\lambda(x-shift)}} \tag{3.8}$$

In the normalisation step, it should be mentioned that if the normalised value is close to one, that means it is a more valuable property, and if the normalised value is close to zero, that means it is a less prioritised property. For example, if the CPU utilisation of the VM is high, the normalisation algorithm should give a value close to zero. We define the following properties for each cloud VM: general VM cost, current cost of the application, workload (i.e. tasks), number of running VMs in the hosting cloud, total number of devices, number of devices that are already connected to a cloud (i.e. active stations), memory size, and CPU cores of VMs. In Table 3.5 we can see the exact values of the normalisation functions.

After the normalisation step, we modify the normalised value to emphasise the importance of the result. This means that if the given x value is greater than our expectation (ν) then we will increase the value of x. The opposite is true when the given x is smaller than ν. To achieve this we will modify the normalised value by using the Kappa function shown in Figure 3.6 with $\nu = 0.4$ and $\lambda = 3.0$ parameters:

$$\kappa_{\nu}^{\lambda}(x) = \frac{1}{1 + \left(\frac{\nu}{1-\nu}\frac{1-x}{x}\right)^{\lambda}} \tag{3.9}$$

Finally, a cloud score number for the given application has to be calculated. To achieve this, we can use conjunction, disjunction, or aggregation operators. The conjunctive operator is similar to the AND operator. This means that if one of the values is small, then the result will also be small. The opposite is true for a disjunctive operator, which is similar to the OR operator. If one of the values is large, the result will also be large. The aggregation operator lies between the disjunctive or conjunctive operator, that is why we use this operator:

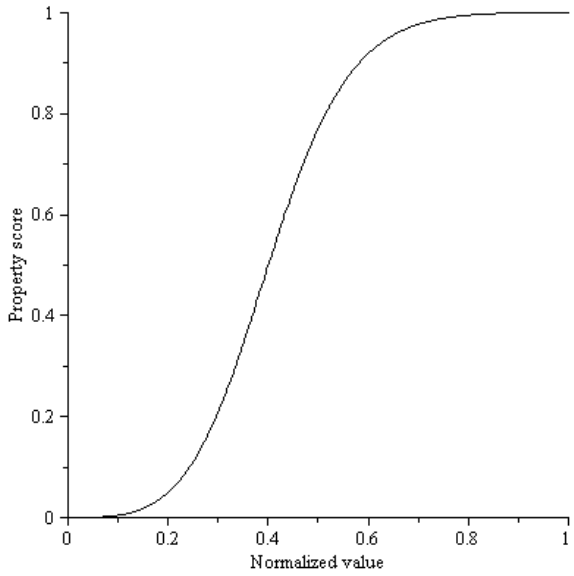


Figure 3.6: *The Kappa function*

$$a_{\nu,\nu_0}(x_1, \cdots, x_n) = \frac{1}{1 + \frac{1-\nu_0}{\nu_0}\frac{\nu}{1-\nu}\prod_{i=1}^{n}\frac{1-x_i}{x_i}}, \tag{3.10}$$

where ν is the neutral value and ν_0 is the threshold value of the corresponding negation. Here we do not want to threshold the result so both parameters have the same value of 0.5. The result of the calculation is always a real number that lies in the [0,1] interval. So we calculate the score for all clouds (i.e. VM queues of clouds) to find which one is the most suitable for a given device. Finally, when the score number is calculated for all clouds, we create a distribution function based on the score number and choose one from this distribution randomly.

3.3.3 Evaluation with Weather Forecasting Scenarios

With our proposed algorithms of the device strategies, we address the optimisation of cloud side-costs with an enhanced allocation of the IoT devices. Which means we can define more cloud providers with their own pricing schemes, but we use only one IoT provider (therefore the IoT-side cost cannot be optimised). To achieve this, we chose to model the crowd-sourced meteorological service of Hungary called Idokep.hu, again. As an enhancement, a sensor's data is not restricted to a standalone cloud service (i.e. application), but multiple choices become available by utilising more clouds.

Table 3.6: *Detailed Bluemix, Azure and Amazon pricing-based private cloud configurations used in the evaluations*

Cloud	Bluemix		
Flavor	Small	Medium	Large
Hourly price (Euro)	0.0378	0.149	0.295
CPU (cores)/RAM (GB)	1/1	4/2	8/4
Cloud	Azure		
Flavor	Small	Medium	Large
Hourly price (Euro)	0.019	0.0579	0.297
CPU (cores)/RAM (GB)	1/1.75	2/3.5	8/14
Cloud	Amazon		
Flavor	Small	Medium	Large
Hourly price (Euro)	0.0229	0.0415	0.3327
CPU (cores)/RAM (GB)	1/2	2/4	8/32

First Scenario

In this scenario, all weather stations have 8 sensors (represented by a device in our model), the message size of the sensors can be set up to 0.05 KBs, and the sensors generate data every minute. The start-up period of the stations was selected randomly between 0 and 20 min. In order to exemplify the usage of different cloud selection strategies, we defined periodic start-up and shut-down dates for certain stations (e.g. to represent malfunctions or failures). We simulate a whole day of operation (from 0:00 to 24:00), and we start the simulation by setting up 200 stations at 0:00 a.m. At 2:00 a.m. we start 100 more and at 10:00 a.m. 200 more to scale the total number of operated stations up to 500. At 2:00 p.m. we shut down 200 stations to scale down the number of running stations to 300 by 10:00 p.m. At the end of the day, the total number of running stations returns to 200. This means the total number of operated meteorological stations in this scenario is 500 (which denotes a relatively small-scale, nationwide system).

With these station management timings we run four different test cases: (i) all stations use the *Random* strategy, (ii) all stations use the *Cost-aware* strategy, then (iii) all stations use the *Load-balanced* strategy. Finally, (iv) we used the *Pliant* strategy in the last experiment. For this evaluation, we configured three clouds based on the *LPDS-1* cloud description from Table 3.7, and every cloud can run application instances (to execute compute tasks) in three VM flavors defined in Table 3.6 (that makes 9 possible application instances in total).

We executed the formerly defined scenario with the four test cases. The results of the experiments can be seen in Table 3.8. After executing this scenario the ap-

Table 3.7: *Detailed multi-cloud configuration for the evaluations*

Cloud	Physical machines
LPDS-1	1 PM - 32 cores, 128 GB RAM 4 PMs - 8 cores, 12 GB RAM
LPDS-2	1 PM - 64 cores, 128 GB RAM 1 PMs - 48 cores, 128 GB RAM 1 PMs - 32 cores, 128 GB RAM 9 PMs - 8 cores, 12 GB RAM
LPDS-3	2 PMs - 64 cores, 128 GB RAM 2 PMs - 48 cores, 128 GB RAM 2 PMs - 32 cores, 128 GB RAM 18 PMs - 8 cores, 12 GB RAM

plications processed 173.75 MBs of data. The so-called timeout parameter denotes how much time it took for the application to terminate (i.e. to perform all remaining data processing operations) after the last station stopped working (at 24:00). As we can see from these results, the cheapest solution is the *Cost-aware* strategy (1.769 Euros) with these simulation parameters, it also has the shortest timeout (1.76 minutes) and it utilises the fewest virtual machines. This strategy only used 5 instances of the cheapest VM while the other strategies used 1 VM instance in every running application. Since the stronger virtual machines (having higher costs) processed the tasks faster than the weaker ones (as expected), they had to generate tasks more frequently. In general, choosing the cheapest VM for an application may result in serious delays in real-time systems, but in this case, our simulated system can operate with weaker resources in real time due to the small amount of sensor data to be processed. At the beginning of the simulation there is no virtual machine running, which can serve any task execution request, thus each simulation has to wait at most 5 minutes to deploy one and allocate tasks, which means all timeout values of the strategies are acceptable.

The *Random*, *Load-balanced*, and *Pliant* strategies use unnecessarily more expensive virtual machines, which results in more than 60 Euros of cost in every case, but we can see the advantage of the *Pliant* strategy: it tries to minimise the timeout value. In this case, it achieved a timeout of 2.06 minutes, which is the second-best result. It shows that our *Pliant* strategy focused more on execution time reduction than cost savings.

Figure 3.7 shows the allocated tasks of an application running in the simulation with the *Pliant* and *Cost-aware* strategies for the first 12 hours. Every box denotes a different task and boxes having the same colour were processed by the same virtual machine. The length of the tasks refers to their execution time. We can see that

Table 3.8: *Evaluation results of the first scenario*

Strategies	Cost-aware	Random	Load-balanced	Pliant
App-1 cost	0	1.119	1.119	1.119
App-2 cost	0	2.027	2.027	2.027
App-3 cost	0	16.167	16.223	16.223
App-4 cost	0	1.842	1.842	1.842
App-5 cost	0	7.300	7.300	7.300
App-6 cost	0	14.426	14.426	14.426
App-7 cost	1.769	0.974	0.972	0.974
App-8 cost	0	2.827	2.822	2.827
App-9 cost	0	14.478	14.454	14.478
Total cost (Euro)	1.769	61.164	61.188	61.219
No. of used VMs	5	9	9	9
Total tasks	227	2,619	2,616	2,619
Timeout (min)	1.76	4.01	4.05	2.06

the tasks of the *Cost-aware* strategy processed a relatively medium amount of data resulting in many, not-too-narrow boxes on the timeline. In case the amount of unprocessed data was growing, the system started to scale up the number of utilised virtual machines. Meanwhile, the *Pliant* strategy worked with stronger and faster virtual machines, which also resulted in many tasks with small amounts of data, but using only the same, best-fit VM. This explains the difference between the number of total tasks of these strategies. Next, we investigate a scenario of a higher scale.

Second Scenario

In the second scenario, we aimed to simulate a larger, worldwide system. An international meteorological service called OpenWeatherMap is operated by the Openweather IT company[17], which was established in 2014 by a group of experts in Big Data and image processing. As their website suggests, they manage over 40,000 meteorological stations all over the world. Our goal with this scenario is to investigate how IoT applications behave in such large-scale environments. Similar to the first scenario, we used three clouds configured with Amazon, Azure, and IBM Bluemix cloud provider pricing defined in Table 3.6, but we modified the physical parameters of the simulated private clouds (to be able to cope with the higher number of stations) as defined by *LPDS-2* in Table 3.7. The number of running weather stations has been increased to 40,000, each of them works with 8 sensors and generates 50 bytes of data every minute. We run this scenario to simulate 6 working hours. In the

[17]OpenWeather (accessed in September, 2018): https://openweather.co.uk/about

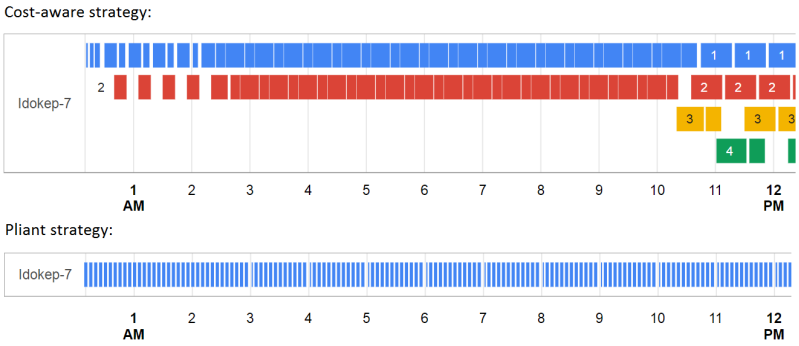


Figure 3.7: *Timeline comparing task allocations of Pliant and Cost-aware strategies in the first scenario*

Table 3.9: *Evaluation results of the second scenario*

Strategies	Cost-aware	Random	Load-balanced	Pliant
App-1 cost	0	3.563	3.544	3.542
App-2 cost	0	3.745	3.707	3.721
App-3 cost	0	12.396	12.451	12.451
App-4 cost	0	5.799	5.796	5.783
App-5 cost	0	9.384	8.324	8.748
App-6 cost	0	12.157	12.132	12.034
App-7 cost	26.419	3.061	3.063	3.090
App-8 cost	0	5.156	5.243	5.166
App-9 cost	0	11.261	11.187	11.112
Total cost (Euro)	26.419	66.527	65.451	65.651
No. of used VMs	109	180	170	173
Total tasks	1,722	1,830	1,819	1,838
Timeout (min)	631	86	86	71

beginning, we started 10,000 stations, then we added 10,000 stations more in the next hours to reach 40,000 stations by the fourth hour.

The results of the second scenario are shown in Table 3.9. After executing this scenario the applications processed 4.008 GBs of data. In the previous scenario the *Cost-aware* strategy was a good choice for both cost and runtime, but problems may occur for systems with higher scale. In this case, the cheapest schedule was provided by the *Cost-aware* strategy with 26.419 Euros, but it had 631 minutes ($\sim$10.51 hours) timeout, which is almost twice longer than the simulated working time (i.e. 6 hours). For applications that are not sensitive to low latency, the *Cost-aware* strategy can still be an acceptable opportunity to decrease costs, but for time-dependent applications

(e.g. smart systems, weather forecasting systems) other strategies are needed. Nevertheless, the *Cost-aware* strategy utilised the lowest number of VMs, too. The *Random* and the *Load-balanced* strategies have the same timeout (86 minutes), but the *Load-balanced* approach operated with fewer virtual machines (10 VMs) and saved around one Euro compared to the *Random* one. The *Pliant* strategy was even better with almost the same price since it reached the most favourable timeout (71 minutes).

Third Scenario

In the third scenario, we configured our private cloud to be the strongest, having twice as many resources as in the second scenario (detailed in the *LPDS-3* parameter setup of Table 3.7), while the rest of the configuration (the applications and the stations) remained untouched, thus the final amount of generated (and processed) data was the same as in the previous scenario.

Table 3.10 shows the results of the third scenario. With the increased physical resources the running time has decreased, but the *Cost-aware* strategy still required 526 minutes (∼8.76 hours) timeout, after the last station stopped working.

Table 3.10: *Evaluation results of the third scenario*

Strategies	Cost-aware	Random	Load-balanced	Pliant
App-1 cost	0	3.512	3.552	3.552
App-2 cost	0	3.724	3.731	3.804
App-3 cost	0	13.283	12.479	12.451
App-4 cost	0	6.233	5.830	5.824
App-5 cost	0	9.197	8.523	8.386
App-6 cost	0	12.428	12.157	11.960
App-7 cost	26.489	3.085	3.071	3.070
App-8 cost	0	5.224	5.185	5.195
App-9 cost	0	11.904	12.251	12.152
Total cost (Euro)	26.489	66.429	66.784	66.397
Used VMs	172	183	184	185
Total tasks	1,722	1,905	1,889	1,893
Timeout (min)	526	36	36	36

If we take a look at the figures, we can see that most strategies benefited from the stronger clouds: they all managed to reduce the timeout significantly. The *Cost-aware* strategy remained the cheapest one, but the number of used virtual machines increased the most against the other strategies compared with the second scenario. The amount of unprocessed data grew faster than the number of available virtual

Table 3.11: *Evaluation results of the fourth scenario*

Strategies	Cost-aware	Load-balanced	Pliant
Total cost (Euro)	10.442	41.765	38.84
Used VMs	81	51	51
Total tasks	685	1,384	1,242
Timeout (min)	41	31	24.9

machines, thus when the application operated with the maximum number of stations the stronger resources could provide more virtual machines to reduce timeout. Comparing the other three strategies, they show minimal deviation in the used virtual machines or the costs. The *Pliant* approach uses the most virtual machines (185), but it was the cheapest with 66.397 Euros. At the same time, all strategies have the same timeout value of 36 minutes. This means that by increasing the number of resources, the strategies behave differently.

Fourth Scenario

In the previous scenarios, the station allocation strategies had to choose only 2-4 times to select VM-queues for the applications processing sensor data of the stations. One of the advantages of the *Pliant* approach is that it is able to take into account more features of the underlying systems, but for this strategy these scenarios were too static, having only a small number of decision points. Thus in the last, fourth scenario, we defined a more dynamic scenario, where we managed 11,500 stations in the following way. Every half an hour, 500 stations started to operate and the whole simulation ran for 12 hours. The *Pliant* algorithm had to decide more often than in the former cases. Our aim with this scenario is to prove that this sophisticated algorithm is able to decrease both the costs and the runtime at the same time. The results can be seen in Figure 3.11. The processed data for the whole experiment is 1.54 GBs. For this scenario, we used a different cloud setup as well. We configured three clouds based on the *LPDS-1*, *LPDS-2*, and *LPDS-3* cloud descriptions of Table 3.7, respectively.

As expected, the cheapest solution is the *Cost-aware* algorithm with 10.442 Euros, which also has the highest timeout of 41 minutes. This strategy used the highest number of virtual machines, which is also a disadvantage if the cloud provider calculates the cost based on the number of VMs. Comparing the other strategies (here we neglected the *Random* approach), the *Pliant* and the *Load-balanced* strategies used the same number of virtual machines, but the *Pliant* algorithm managed to reduce both the cost and the runtime most effectively.

3.4 Discussion and Concluding Remarks

Distributed system simulators are not generic enough to be applied in newly emerging domains, such as IoT-Cloud systems, which require in-depth analysis of the interaction between IoT devices and cloud resources. Therefore, in this chapter, we introduced a method to show how generic IoT sensors could be modelled in a state-of-the-art cloud simulator.

We also analysed the operating costs of IoT applications by introducing a model of pricing schemes of four providers, and then comparing them by calculating the costs of a real-world meteorological application.

Finally, we introduced four cloud selection strategies aimed to reduce IoT application execution time and usage costs. Our presented, dynamic approach based on the Pliant method can adapt to the actual state of the underlying, possibly multi-cloud systems, therefore, it can find better placement of devices, resulting in lower costs and response times.

The results of this chapter belong to **Thesis II**, and its contents were published in papers [P1], [P2], [P5], [P6], and [P10]. My contributions presented in this chapter are the following:

II/1. I investigated IoT-Cloud use cases to derive a general IoT use case based on meteorological forecasting, and I used it to evaluate the proposed IoT model.

II/2. I laid the foundations for the flexible and scalable modelling of IoT systems, and I implemented it in the DISSECT-CF-IoT simulator.

II/3. I analysed the operating costs of the meteorological IoT use case, and I developed a novel cost estimation extension using real cloud and IoT provider pricing schemes.

II/4. I introduced greedy and Pliant-based resource allocation strategies to reduce application execution time and utilisation costs for multi-cloud environments.

4

Simulating IoT Systems in a Multi-layered Fog Environment

4.1 Introduction

The Internet of Things (IoT) paradigm forms sensors, actuators, and smart devices into a complex system via the Internet. IoT is often supported by Cloud Computing because the huge amount of sensed data requires elastic storage and processing services for further analysis. In the past years, a new paradigm called Fog Computing has grown out of Cloud Computing, where the generated sensor data are stored and processed on so-called fog nodes, which are located geographically closer to the end-users for minimising latency and ensuring the privacy of the data [39].

A typical fog topology is shown in Figure 4.1, where sensors and actuators of IoT devices are located at the lowest layer. Based on their configuration and type, things produce various raw sensor data. Sensors are mostly resource-constrained and passive entities with restricted network connection, on the other hand, actuators ensure broad functionality with an Internet connection and enhanced resource capacity [48]. They aspire to make various types of decisions by assessing the processed data retrieved from the nodes. These actions can affect the physical environment or refine the configuration of the sensors. Furthermore, the embedded actuators can manipulate the behaviour of smart devices, for instance, restart or shut down a device, and motion-related responses can also be triggered.

In the surrounding world of IoT devices, location is often fixed, however, the

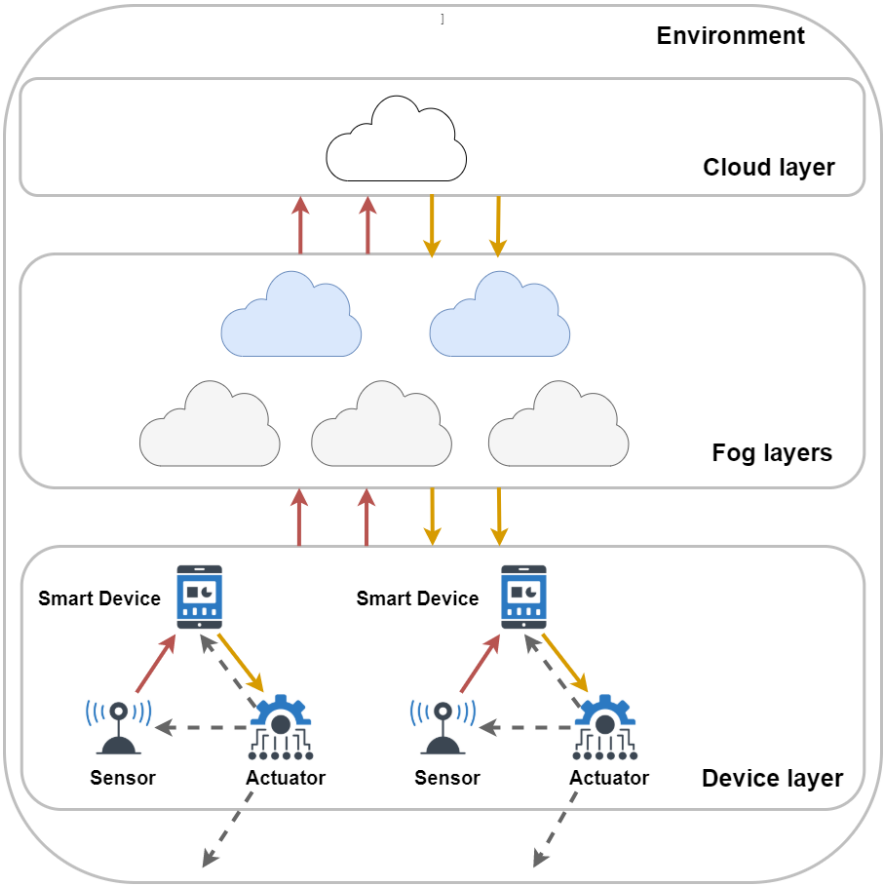


Figure 4.1: *The connections and layers of a typical fog topology*

Quality of Service (QoS) of these systems should also be provided at the same level in the case of dynamic and moving devices. Systems composed of IoT devices supporting mobility features are also known as the Internet of Mobile Things (IoMT) [46]. Mobility can have a negative effect on the QoS to be ensured by fog systems, for instance, they could increase the delay between the device and the actual node it is connected to. Furthermore, using purely cloud services can limit the support for mobility [53].

Besides scalability, latency, and resource management issues, the energy consumption of a fog environment and the corresponding smart devices is also a great challenge as stated in [4], therefore, it should be considered as one of the key factors for the development of Fog Computing solutions. Energy-efficient solutions also have a significant impact on the carbon footprint and on climate change. In order to avoid wasting energy, smart decisions could take into account IoT device motion or corresponding environmental parameters, in order to optimally handle the related equipment.

The remainder of this chapter is as follows: Section 4.2 extends the related works

of IoT-Fog-Cloud systems towards modelling actuators and mobility. Next, Section 4.3 introduces another type of strategy to handle task offloading decisions of the multi-layered fog topology. Section 4.4 then presents the detailed model for simulating actuator and mobility use cases, and Section 4.5 introduces the energy-aware extension of the simulation environment. Finally, 4.6 concludes the chapter with the contributions.

4.2 Related Works

A detailed comparison of the related simulation approaches and the main components of fog modelling in DISSECT-CF-Fog were already presented in the first chapter (see Section 2.3 and 2.4), but to support the modelling of the latest technological enhancements in the addressed research fields, we had to extend our literature review towards actuator modelling and mobility simulation.

The CloudSim-based iFogSim simulator [23] is one of the leading fog simulators within the research community, which follows the sense-process-actuate model. In this tool, the actuator is declared as the responsible entity for the system or a mechanism, and the actualisation event is triggered when a task, which is known as a Tuple, determining a certain amount of instruction and size in bytes, is received by the actuator. In the current implementation of iFogSim, this action has no significant effect, however, custom events can also be defined by overriding the corresponding method. Nevertheless, no such events are created by default. The actuator component is determined by its connection and network latency. The original version of iFogSim does not support mobility, however, the static, geographical location of a node is stored.

Another CloudSim extension is EdgeCloudSim [64], which aims to ensure mobility support in simulation environments. It associates the position information of a mobile device with a two-dimensional coordinate point, which can be updated dynamically. This simulation solution considers the nomadic mobility model, by the definition of which a group of nodes moves randomly from one position to another. This work also takes into account the attractiveness of a position to define the duration of stay at some place. Further mobility models can be created by extending the default class for mobility, but there is no actuator entity implemented in this approach.

FogNetSim++ [56] can be used to model fog networks supporting heterogeneous devices, resource scheduling, and mobility. In this paper, six mobility strategies were proposed, and new mobility policies can also be added. This simulator aids the entity mobility models, which handle the nodes independently, and take into account parameters such as speed, acceleration, and direction in a three-dimensional coordinate system. Unfortunately, the source code of the simulator presents examples of

the linear and circular mobility behaviour only. This simulation tool used no actuator model.

YAFS [36] is a simulator that analyses IoT application deployments and mobile IoT scenarios. The actuator in this realisation is defined as an entity, which receives messages with the given number of instructions and bytes, similar to the solution of iFogSim. The paper also mentioned dynamic user mobility, which takes into account different routes using GPX formats (it is used by applications to depict data on the map), but this behaviour was not explained or experimented with.

[30] proposed the IoTSim-Edge simulation framework by extending the CloudSim model towards IoT and Edge systems. This simulator focuses on resource provisioning for IoT applications considering the mobility function and battery usage of IoT devices, and different communication and messaging protocols as well. The IoTSim-Edge contains no dedicated class for the actuator components, nevertheless, the representative class of an IoT device has a method for actuator events, which can also be overridden. There is only one predefined actuator event affecting the battery of an IoT device, however, it was not considered during the evaluation phase by the authors. This simulation tool also takes into consideration the mobility of smart devices. The location of a device is represented by a three-dimensional coordinate system. Motion is influenced by a given velocity and range, where the corresponding device can move, and only horizontal movements are considered within the range by the default moving policy.

MobFogSim [54] aims to model user mobility and service migration, and it is one of the latest extensions of the iFogSim, where actuators are supported by default. Furthermore, the actuator model was revised and improved to handle migration decisions, because migration is often affected by end-user motions. To represent mobility, it uses a two-dimensional coordinate system, the users' direction and velocity. The authors considered real datasets as mobility patterns, which describe buses and routes of public transportation.

The actuator and mobility abilities of these simulators are further detailed in Table 4.1. The second column shows possible directions for transferring the sensor data (usually in the form of messages), in case the actuator interface is realised in the corresponding simulator. It can be observed that it basically follows similar logic in all cases. The third column highlights actuator events that can be triggered in a simulator. The fourth column shows the supported mobility options (we only listed the ones offered in their source code) and finally, we denote the position representation manner in the last column.

It can be observed that there is a significant connection between mobility support and actuator functions, but only half of the investigated simulators applied both of them. Since the actuator has no commonly used software model within the latest simulation tools, developers omit it, or it is left to the users to implement it, which

Table 4.1: *Detailed characteristics of the related simulation tools*

Simulator	Communication direction	Actuator events	Mobility	Position
DISSECT-CF-Fog *(this work)*	- Sensor → Fog / Cloud → Actuator - Sensor → Actuator	- 10 different predefined actions for actuation - Adding new by overriding	- Nomadic - Random Walk	Latitude, Longitude
iFogSim	- Sensor → Fog → Actuator	- Default, but it can be overridden	-	Coordinates
EdgeCloudSim	-	-	- Nomadic	Coordinates
FogNetSim++	-	-	- Linear - Circular	Coordinates
IoTSim-Edge	- Sensor → Fog Device → Actuator	- Default, but it can be overridden	- Linear	Coordinates
YAFS	- Sensor → Service → Actuator	-	- Real dataset	Latitude, Longitude
MobFogSim	- Mobile Sensor → Mobile Device → - Mobile Actuator	- Migration	- Linear - Real dataset	Coordinates

can be time-consuming (considering the need for additional validation). In a few cases, both actuator and mobility models are simplified or just rudimentary handled, thus realistic simulations cannot be performed.

4.3 Managing Offloading Decisions in DISSECT-CF-Fog

In order to serve actuator and mobility decisions, first we have to introduce an offloading mechanism for IoT applications, because if the selected fog node is overloaded, the execution of the appropriate task will be delayed, which has a negative effect on the makespan of its application and/or on the execution costs. To reach the required Quality of Service of an IoT application, the management of IoT-Fog-Cloud systems should also take into account the position and the actual load of fog nodes.

When the number of tasks is growing, a single fog node may not be able to process them continuously, therefore, a forwarding function for some of the tasks to other nodes can be useful to manage a higher number of tasks of an IoT application. A fog topology consisting of several nodes with different locations can handle the unforeseen appearance of smart devices (and new tasks) more effectively, than a single, heavyweight cloud node. Such functions can take into account the physical location of the entities of the execution environment (i.e. fog or cloud node or IoT devices), the load of the network used for communication and data transfers, and the transfer, storage, and execution costs.

To overcome this problem, multi-layer fog node management was introduced in DISSECT-CF-Fog by enabling task offloading from (possibly overloaded) nodes to others. A typical fog topology can contain numerous nodes, some of them are grouped as a fog cluster, which restricts the access and visibility of other nodes. These nodes can be ordered into layers, where higher-level fog layers usually contain stronger phys-

ical resources. The computing nodes can be heterogeneous and often restricted on certain types and strengths of VMs and applications in order to serve as many users as possible, such restrictions usually concern the processing power and the memory usage of the given VM.

To manage the offloading decisions separately for each node, we introduce the application strategy components with which different task allocation approaches can be created and implemented taking into consideration the characteristics of the topology. In our experiments, we also exploit the recently introduced capabilities and components of the simulator. The parallel steps of the simulation are the following, detailed briefly: (i) sensors of smart devices generate data in the given frequency, (ii) the unprocessed data are forwarded to a node (based on the applied device strategy presented in Chapter 3), (iii) the data are packaged in a compute task, (iv) VMs on a node process as many tasks as they can, and finally (v) if a node is overloaded, then the unprocessed tasks will be forwarded to another node.

4.3.1 The Proposed Task Allocation Strategies for Fog Nodes

We defined four basic strategies for task allocation to validate the usability of our proposal. The *Random* strategy is the default, which always chooses one from the connected nodes randomly. The *Push Up* strategy always chooses the connected parent node (i.e. a node from a higher layer), if available. This approach does not take into account the properties of the neighbours, and basically ensures the fastest way to forward unprocessed tasks to the cloud, where more powerful VMs may reduce the processing time of it. The disadvantage of these strategies is the disability to consider increased network traffic and costs of the operation in the decision. The third strategy called *Hold Down* aims to address privacy needs because the system can keep application data as close to the end-user as possible. In this way, the network traffic is minimal, but the execution time of the application can increase dramatically (due to the possible overload of constrained resources at the lowest layer). The *Runtime-aware* strategy ranks the available parent nodes, and all neighbour nodes (from its own layer) by network latency and by the ratio of the available CPU capacity and the total CPU capacity. The algorithm picks the node with the highest rank (i.e. the closest and least loaded one). The last strategy we propose is an algorithm that can predict which computing node could be the best for managing a given IoT device (according to the actual state of the system, represented by its properties). This algorithm is also based on the Pliant logic (as it is described in Section 3.3.2), therefore, for each reachable fog and cloud node the *Pliant* strategy calculates a score number using normalisation, Sigmoid and Kappa functions, and the aggregation operator to choose the appropriate one for the offloading decision. We define the following three properties for each system node: load, cost, and unprocessed data of a node. In Table

Table 4.2: *Normalisation parameters*

Parameter	Lambda	Shift
Workload of a node	-1.0/8.0	(Maximum workload - minimum workload) / 2
Price of using a node	4.0	Minimum price
Unprocessed data	-1.0/4.0	(Maximum unprocessed data - minimum unprocessed data)

4.2, we can see the exact values of the normalisation functions.

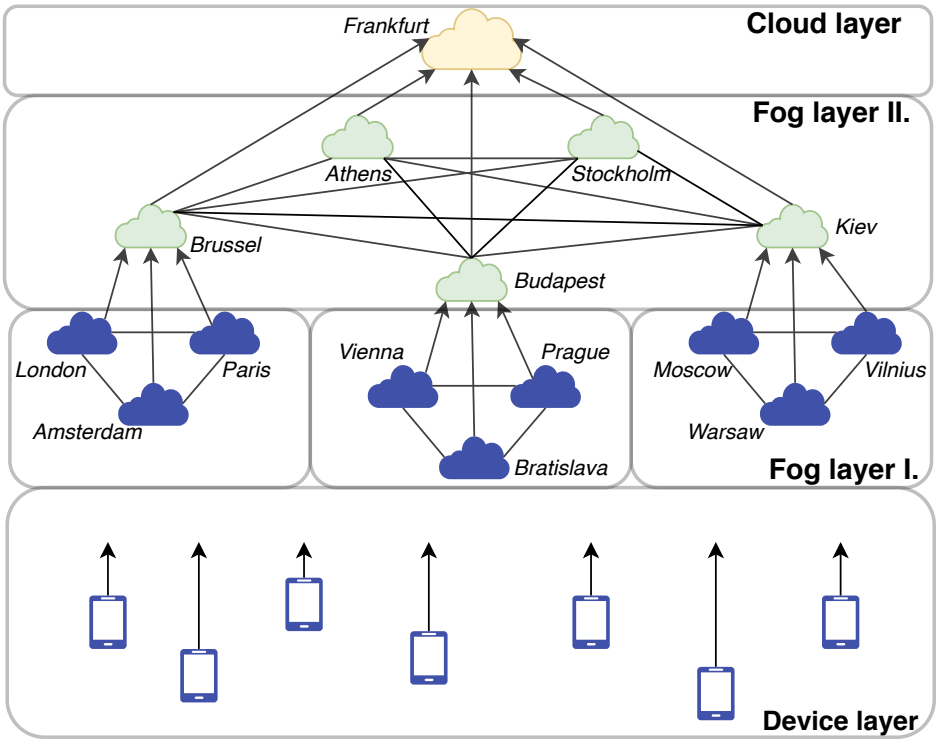


Figure 4.2: *The considered fog topology in the evaluation*

4.3.2 The Considered Scenarios and Their Configuration

The stakeholders of IoT applications and resource providers have to prepare in advance for the unforeseen data generated by sensors and often aspire to treat them in real-time, thus a fog topology should be robust enough and be able to handle a vast amount of IoT devices. Such requirements can be investigated by defining application and device strategies. Thus, we applied the device strategies for our experiments, which were introduced in Section 3.3, but some revisions were also required. As a reminder, a device strategy is applicable to map any device to any preferred node, according to the policy of the actual device. Similar to other models, resource needs,

Table 4.3: *Evaluation results of the first scenario*

First scenario					
Device strategy	Distance-based				
Application strategy	Pliant	Runtime-aware	Random	Hold Down	Push Up
Num. of VMs	75	73	75	56	70
Cost ($)	76.1	73.6	70.4	57.0	81.7
Network utilisation (sec.)	31	41	44	0	39
Data transferred (MB)	101	144	176	0	140
Timeout (min.)	**4.6**	4.9	6.0	228.5	4.9
Device strategy	Load-balanced				
Application strategy	Pliant	Runtime-aware	Random	Hold Down	Push Up
Num. of VMs	72	72	72	36	46
Cost ($)	118.3	122.3	117.7	81.9	112.1
Network utilisation (sec.)	87	147	102	0	21
Data transferred (MB)	4,057	9,560	4,851	0	2,654
Timeout (min.)	**27.8**	70.3	57.8	8,842.0	1,529.8
Device strategy	Mixed				
Application strategy	Pliant	Runtime-aware	Random	Hold Down	Push Up
Num. of VMs	77	77	77	41	54
Cost ($)	93.7	99.4	91.5	59.3	87.1
Network utilisation (sec.)	60	95	76	0	34
Data transferred (MB)	1,414	2,361	1,766	0	1,197
Timeout (min.)	11.6	30.9	14.8	4,371.0	**8.5**

energy consumption, latency, bandwidth, and IoT/cloud side-cost can be considered, in order to ensure appropriate pricing or to reduce IoT application execution time (i.e. makespan).

The *Distance-based* device strategy chooses the geographically closest fog node, to be selected only from the lowest fog layer, for a device to communicate with (i.e. send tasks to). The disadvantage of this strategy lies in data overload in a certain node if many devices were located in its neighbourhood. The *Load-balanced* device strategy always tries to balance the number of connected devices for each node considering the number of available physical machines of the node as well. The disadvantage of this strategy is the increased latency if a device chose a farther-located node from the lowest layer. One can observe that considering only device strategies can cause a bottleneck effect in the topology.

We evaluated the proposed device mapping and task allocation strategies with three different scenarios simulating a European-wide weather forecasting system. The first scenario deals with 5,000 devices (i.e. weather stations). In the second sce-

Table 4.4: *Evaluation results of the second scenario*

Second scenario					
Device strategy	Distance-based				
Application strategy	Pliant	Runtime-aware	Random	Hold Down	Push Up
Num. of VMs	90	90	90	59	79
Cost ($)	149.2	150.0	147.9	106.0	138.4
Network utilisation (sec.)	150	214	189	0	111
Data transferred (MB)	3,835	8,041	5,483	0	2,103
Timeout (min.)	**24.0**	50.5	24.5	1,918.5	349.8
Device strategy	Load-balanced				
Application strategy	Pliant	Runtime-aware	Random	Hold Down	Push Up
Num. of VMs	72	60	72	36	46
Cost ($)	191.9	170.8	192.3	134.4	195.9
Network utilisation (sec.)	177	1,305	210	0	35
Data transferred (MB)	14,700	20,636	16,726	0	6,011
Timeout (min.)	2,436.9	**742.3**	2,464.8	19,133.5	5,634.8
Device strategy	Mixed				
Application strategy	Pliant	Runtime-aware	Random	Hold Down	Push Up
Num. of VMs	87	86	87	56	71
Cost ($)	148.4	155.8	151.4	108.4	152.3
Network utilisation (sec.)	144	332	185	0	58
Data transferred (MB)	6,476	31,894	8,077	0	3,147
Timeout (min.)	**99.8**	182.3	174.8	10,143.5	1,962.3

nario, we doubled the number of devices, hence 10,000 stations utilised the system. In both cases, the whole simulation period took one day. Finally, in the last scenario, we modelled a more dynamic system. In the beginning, only 2,000 stations started to work, and after every 4 hours we installed 2,000 more, thus at the end of the day, the total number of devices was 12,000. In all cases, each station was equipped with five sensors (e.g. measuring weather conditions, such as temperature and humidity, with 50 bytes of sensor data). The time interval between two measurements (i.e. the data generation frequency) was set to one minute.

The topology contains two fog layers with 14 different fog nodes organised into clusters, and one cloud layer having a single cloud node. Each node is represented by a different city and the latency between them is measured by using WonderNetwork[18]. The defined topology can be seen in Figure 4.2, which also depicts the fog clusters and layers with different colours. The arrows represent routes which are responsible for communication between the layers, while the (undirected) edges are

[18]WonderNetwork (accessed in May, 2020): https://wondernetwork.com/pings

for the message exchanges inside a cluster. The network capability of the smart devices (or stations) is modelled with a 4G network with an average 50 ms of latency. When a connection is created by applying a device policy, the exact latency value will be weighted in proportion to the physical distance between the device and the node.

The nodes of the topology are modelled with real VM specifications and pricing schemes according to the Amazon Web Services (AWS): the lowest Fog layer has VMs with 2 CPU cores, 4 GB RAM and 0.051$ hourly price, the top fog layer has VMs with 4 CPU cores, 8 GB RAM and 0.101$ hourly price, finally, the cloud layer has VMs with 8 CPU cores, 12 GB RAM and 0.204$ hourly price. Each VM can process only one task (represented by 250 KB of data) at a time. The lowest fog layer of the topology was divided into three clusters, one node of the cluster tackled with 12 CPU cores and 24 GB RAM altogether. We doubled the resource capacity of a node of the upper fog layer, thus it dealt with 24 CPU cores and 48 GB RAM.

Our preliminary evaluation showed that applying this scheme made the topology (the system) particularly strong, therefore, some strategies (e.g. *Push Up*) became more beneficial than others. Therefore, the cloud resources were set to 48 GB CPU cores and 96 GB RAM.

Concerning the application management in the simulator during executing a scenario, the ratio of the forwardable data is limited to 50% (hence moving all data to a different node is prohibited). The daemon service of the application decides after every 150 seconds about the task allocation request to a VM of a node. In each scenario, the locations of all devices were fixed during the simulation, however, those positions were randomly chosen at the beginning of the simulation to enable realistic and unexpected behaviour of such devices. The start time of the devices had a delay randomly set from the 0-20 interval in minutes (to avoid burst operations, and also to be more realistic).

Evaluation of the Strategies

Table 4.3, Table 4.4, and Table 4.5 summarise the average results after executing the scenarios with all device and application strategies three times. Table 4.6 presents the mean of the results of the scenarios. We also extended our measurements with an extra device strategy called *Mixed*, where half of the IoT devices were managed with the *Distance-based*, and the other half of the devices were managed with the *Load-balanced* strategy.

To compare the proposed strategies, we measured how many VMs were required for processing the tasks (and their data) during operating hours. The Network utilisation metric reflects the network load, and it represents the time taken to transfer the sensor data from the source node to the actual processing node. The Data transferred metric represents the total size of all forwarded data. The Data transferred and the Network utilisation metrics were rounded to the nearest integer number.

Table 4.5: *Evaluation results of the third scenario*

Third scenario					
Device strategy	Distance-based				
Application strategy	Pliant	Runtime-aware	Random	Hold Down	Push Up
Num. of VMs	90	90	90	60	79
Cost ($)	110.1	111.9	107.8	77.1	105.0
Network utilisation (sec.)	87	131	110	0	69
Data transferred (MB)	2,117	5,471	2,939	0	1,099
Timeout (min.)	**74.8**	144.4	111.6	1,091.0	332.3
Device strategy	Load-balanced				
Application strategy	Pliant	Runtime-aware	Random	Hold Down	Push Up
Num. of VMs	90	90	90	51	66
Cost ($)	132.8	138.1	131.5	106.9	134.8
Network utilisation (sec.)	114	204	140	0	40
Data transferred (MB)	4,307	13,599	5,300	0	2,911
Timeout (min.)	195.3	**172.8**	217.8	3,095.0	1,852.3
Device strategy	Mixed				
Application strategy	Pliant	Runtime-aware	Random	Hold Down	Push Up
Num. of VMs	90	90	90	60	80
Cost ($)	111.0	112.8	110.2	78.0	107.3
Network utilisation (sec.)	83	121	108	0	55
Data transferred (MB)	2,194	5,126	3,402	0	1,251
Timeout (min.)	**68.7**	112.7	72.3	1,641.0	254.8

The Timeout value means the time taken to finish data processing after the last sensor measurement was performed. This metric relates to the makespan of an application. Here, the less time required to process all tasks of an IoT application (after stopping the devices), the better a node selection strategy is. According to the used AWS pricing models, we could calculate the exact costs of the usage of the fog and cloud resources (after executing the IoT applications).

Concerning the results of the first scenario, the *Pliant* strategy manages the task with the best Timeout value in the first test case, however, we can save money (73.6$) with the *Runtime-aware* application strategy which has only slightly worse performance with 4.9 minutes. It is obvious that with this topology the lowest layer is inadequate for fast data processing, but it ensures the cheapest solutions in each test case with the *Hold Down* policy. When the smart devices applied the *Load-balanced* device strategy, it increased all values heavily, but still, the *Pliant* algorithm processed the tasks in the fastest way with 27.8$. In the third test we can see that the processing of all generated tasks is cheaper (with 87.1$) and faster (with 8.5 minutes) in

Table 4.6: *The mean of the results of the scenarios*

	First scenario - average values		
Device strategy	Distance-based	Load-balanced	Mixed
Num. of VMs	69.8	**59.6**	65.2
Cost ($)	**71.8**	110.5	86.2
Network utilisation (sec.)	**31.0**	71.4	53.0
Data transferred (MB)	**112.2**	4,224.4	1,347.6
Timeout (min.)	**49.8**	2,105.5	887.4
	Second scenario - average values		
Device strategy	Distance-based	Load-balanced	Mixed
Num. of VMs	81.6	**57.2**	77.4
Cost ($)	**138.3**	177.0	143.2
Network utilisation (sec.)	**132.8**	345.4	143.8
Data transferred (MB)	**3,892.4**	11,614.6	9,918.8
Timeout (min.)	**473.4**	6,082.4	2,512.5
	Third scenario - average values		
Device strategy	Distance-based	Load-balanced	Mixed
Num. of VMs	81.8	**77.4**	82.0
Cost ($)	**102.4**	128.8	103.9
Network utilisation (sec.)	79.4	99.6	**73.4**
Data transferred (MB)	**2,325.2**	5,224.6	2,394.6
Timeout (min.)	**350.8**	1,106.6	429.9

the cloud (*Push Up*). According to these results, it seems the IoT application handles the devices better, if they run with the *Distance-based* strategy.

In the second scenario, we investigated how our IoT application behaves if it has to deal with a doubled number of smart devices. In the first and the third test cases, the *Pliant* handles the increased data the best with 24.0 minutes and 99.8 minutes, respectively. However, applying the *Load-balanced* device strategy with *Runtime-aware* application policy reaches better results. These results show that the distance between the devices and the nodes has serious effects on the IoT application execution, and similar to the first scenario, the application strategies cannot handle the impact of the *Load-balanced* devices.

Concerning the results of the third scenario, two of the device strategies (*Distance-based, Mixed*), and the *Plant* application strategy managed the best Timeout values with 74.8$ and 68.7$. Similar to the second scenario, an IoT application only handles the data in a fair way when we applied the *Load-balanced* strategy with the *Runtime-*

aware strategy.

Concerning the average results for the device strategy, we can see that the best strategy for the operation cost, the network transfer and the timeout value is the *Distance-based* device strategy. In one case, the *Mixed* had a good influence on the network utilisation of the third scenario, which means location independence of the smart devices may require further investigation. We can clearly see that the *Load-balanced* strategy dealt with the least number of virtual machines and it became the best scenario, but it also dramatically increased the rest of the metric values.

4.4 The Actuator and Mobility Models of DISSECT-CF-Fog

In the layered architecture of IoT, actuators are located in the perception layer, which is often referred to as the lowest or physical layer that requires the most detailed level of abstraction in IoT. The actuator interface should facilitate a more dynamic device layer and a volatile environment in a simulation. Therefore, it is preferred to be able to implement actuator components in any kind of simulation scenario, if needed. In our model, one actuator is connected to one IoT device for two reasons in particular: (i) it is observing the environment of the smart device and can act based on previously specified conditions, or (ii) it can influence some low-level sensor behaviour, for instance, it changes the sampling interval of a sensor, resets or completely stops the smart device.

The latter indirectly conveys the conception of a reinterpreted actuator functionality for simulator solutions. The DISSECT-CF-Fog actuator can also behave as a low-level software component for sensor devices, which makes the model compound. The actuator model of DISSECT-CF-Fog can only operate with compact, well-defined events, that specify the exact influence on the environment or the sensor. The set of predefined events during a simulation provides a restriction to the capability of the actuator and limits its scope to certain actions that are created by the user or already exist in the simulator. A brief illustration of sensor-based events is shown in Figure 4.3.

The determination of the exact event, executed by the actuator, happens in a separate, reusable, and extendable logic component. This logic component can serve as an actual actuator configuration, but can also be used as a descriptor for environmental changes and their relations to specific actuator events. This characteristic makes the actuator interface thoroughly flexible and adds some more realistic factors to certain simulation scenarios. With the help of the logic component, the actuator interface works in an automatic manner. After a cloud or fog node has processed the data generated by the sensors, it sends a response message back to the actuator, which

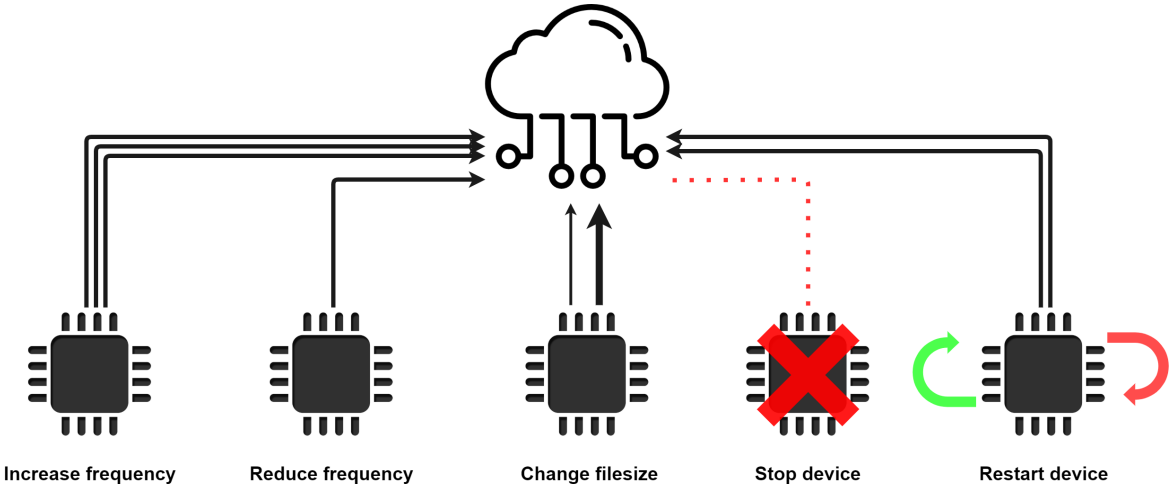


Figure 4.3: *Low-level sensor events*

chooses an action to be executed. This models the typical sensor-service-actuator communication direction.

Unexpected actions may occur in real-life environments, which are hard for algorithms to define, and the execution of some events may not require cloud or fog processes, e.g. when a sensor fails. To be able to handle such issues, the actuator component is capable of executing events apart from its predefined configuration. This feature facilitates immediate and direct communication between sensors and actuators through a smart device.

The proximity of computing nodes is the main principle of Fog Computing and it has numerous benefits, but mobile IoT devices may violate this criterion. These devices can move further away from their processing units, causing higher and unpredictable latency. When a mobile device moves out of range of the currently connected fog node, a new, suitable fog node must be provided. Otherwise, the quality of service would drastically deteriorate and due to the increased latencies, the fog and cloud nodes would hardly be distinguishable in this regard, resulting in losing the benefits of Fog Computing.

Another possible problem that comes with mobile devices is service migration. The service migration problem can be considered as when, where and how (W2H) questions. Service migration usually happens between two computing nodes, but if there is no fog node in an acceptable range, the service could be migrated to the smart device itself, causing lower performance and shorter battery time.

The physical location of fog nodes in a mobile environment is a major concern. Placing Fog Computing nodes too far from each other will result in higher latency or connection problems. In this case, IoT devices are unable to forward their data, hence they are never processed. Some devices may store their data temporarily until they

connect to a fog node, but this contradicts the real-time data processing promises of fogs. A slightly better approach would be to install fog nodes fairly dense in space to avoid the problem discussed above. However, there might be some unnecessary nodes in the system, causing a surplus in the infrastructure, which results in resource wastage.

Considering different mobility models for mobile networks in simulation environments have been researched for a while. The survey by [12] presents 7 entity and 6 group mobility models in order to replace trace files, which can be considered as the footprints of movements in the real world. Applying mobility models is a reasonable decision because they mimic the movements of IoT devices in a realistic way. The advent of IoT and the technological revolution of smartphones have brought the need for seamless and real-time services, which may require an appropriate simulation tool to develop and test the cooperation of Fog Computing and moving mobile devices.

The current extension of the DISSECT-CF-Fog was designed to create a precise geographical position representation of computing nodes (fog, cloud) and mobile devices and simulate the movements of devices based on specified mobility policies. As the continuous movement of these devices could cause connection problems we consider the following events shown in Figure 4.4.

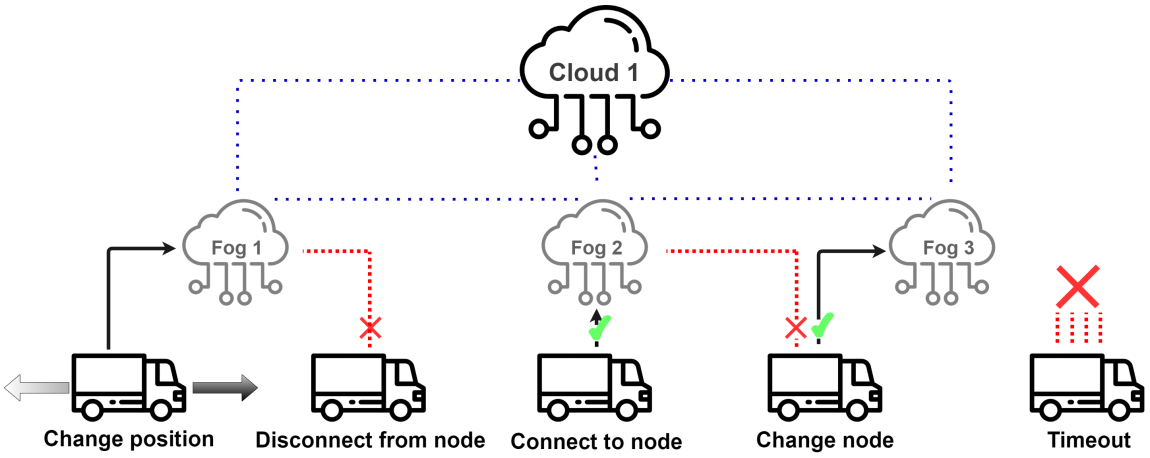


Figure 4.4: *Actuator events related to mobility behaviour*

Examining the occurrence of these specific actuator and mobility events can help in optimising the physical allocation of fog nodes depending on the mobility features of IoT devices.

4.4.1 Actuator Implementation in DISSECT-CF-Fog

For the proper behaviour of the actuator, the data representation in the simulator needs to be more detailed and comprehensive. Consequently, this extension of the DISSECT-CF-Fog simulator introduces a new type of data fragment in the system, to store specific details throughout the life-cycle of the sensor-generated data. Such fragments are forwarded to an IoT application located in a fog or cloud node to be processed. The new IoT data representation handles the following attributes and information:

- *source*: Holds a reference to the IoT device generating sensor data, so the system keeps track of the data source.
- *destination*: Holds a reference to the IoT application of a fog node where the data has been originally forwarded to.
- *data path*: In some cases, fog nodes cannot process the current data fragment, therefore, they might send it to another one. This parameter keeps track of the visited fog nodes by the data before it has been processed.
- *event size*: The size of the response message sent from a fog node to the actuator component. This helps to simulate network usage while sending information back to the actuator.
- *actuation needed*: Not every message from the IoT device requires an actuator response event. This logical value holds true if the actuator should take action after the data has been processed, otherwise it is false.
- *fog needed*: A logical value is true if the data must be processed in a fog node, and should not be sent to the cloud. It is generally set to true when real-time response is needed from the fog node.
- *start time*: The exact time in the simulator when the data was generated.
- *process time*: The exact time in the simulator when the data was processed.
- *end time*: The exact time in the simulator when the response has been received by the actuator.
- *event type*: This is the specific event type that is sent back to the actuator for execution.
- *MTTF*: The mean time until the sensor fails. This attribute is essential to calculate the sensor's average life expectancy, which helps in modelling sensor failure events. If the simulation's time exceeds the *MTTF* value, the sensor has a higher chance to fail.

- *maximum latency*: Its value determines the maximum latency tolerated by the device when communicating with a computing node. For instance, in the case of medical devices, this value is generally lower than in the case of agricultural sensors. Mobile devices may move away from fog nodes inducing latency fluctuations and this attribute helps to determine whether a computing node is suitable for the device, or the expected latency exceeds this maximum latency limitation, therefore the device should look for a new computing node. This attribute plays a major role in triggering fog-selection actuator events when the IoT device is moving between fog nodes.

When creating an IoT device in the simulation, its sensors start generating the new kind of data segments, which later are forwarded to a certain IoT application of a fog node, based on the fog selection strategy of the device. If the actual fog node has adequate resources to process the received data, the processing happens, and if the *actuation needed* attribute of the processed data object was true, then it is sent back to the actuator (i.e. the data source) expanded with a specific actuator event object denoting an action to be performed by the actuator. Otherwise, if the current fog node does not have the capacity to process the data, it sends them over to another node based on the actual strategy of the application.

As mentioned, the actuator model must only operate with predefined events to limit its scope to certain actions. These events are represented by an interface and should be implemented in order to specify an exact action. There are some predefined events in the system: five of them are low-level, sensor-related events, and the other five are related to the mobile functionality of the devices, but these can be extended to different types of behaviours.

Since the actuator has the ability to control the sensing process itself [53], half of the predefined actuator events foster low-level sensor interactions. The *Change file size* event can modify the size of the data to be generated by the sensor. Such behaviour reflects use cases, when more or less detailed data are required for the corresponding IoT application, or the data should be encrypted or compressed for some reason. *Increase frequency* and *Decrease frequency* might be useful when the IoT application requires an increased time interval between the measurements of a sensor. A typical use case of this behaviour is when a smart traffic control system of a smart city monitors the traffic at night when usually fewer inhabitants are located outside. The *Decrease frequency* is the opposite of the previously mentioned one, a typical procedure may appear in IoT healthcare, for instance, the blood pressure sensor of a patient measures continuously increasing values, thus more frequent perceptions are required. The *Stop device* event imposes a fatal error on a device, typically occurring randomly, and it is strongly related to the *MTTF* value described above. The *MTTF* is considered as a threshold, before reaching it, there is only a small chance for failure, after exceeding it, the chance of a failure increases expo-

nentially. Finally, the *Restart device* reboots the given device to simulate software errors or updates.

Customised events can also be added to the simulation by overriding the related methods, which describe the series of actions to occur upon executing the event. In order to model a broad spectrum of scenarios as detailed as possible, we introduce a third type of strategy. The actuator strategy makes it possible to represent an environment around an IoT device, and make the actuator component reactive to its changes by selecting the corresponding event. For instance, let us consider a humidity sensor and a possible implementation of the actuator component. We can then mimic an agricultural environment in the actuator strategy with the help of some well-defined conditions to react to changes in humidity values, and select the appropriate customised actuator events (e.g. opening windows, or watering), accordingly. This characteristic enables DISSECT-CF-Fog to simulate environment-specific scenarios while maintaining its extensive and generic feature.

4.4.2 Representing IoMT Environments in DISSECT-CF-Fog

The basis of mobility implementations in the competing tools usually represent the position of users or devices as two or three-dimensional coordinate points, and the distance between any two points is calculated by the Euclidean distance, whereby the results can be slightly inaccurate. To overcome this issue and have a precise model, we take into account the physical position of the end users, IoT devices and data centres (fog, cloud) by longitude and latitude values. The representative class calculates distance using the Haversine formula [74]. Furthermore, applying geographical location with a coordinate system often results in a restricted map, where the entities are able to move, thus in our case worldwide use cases can be implemented and modelled.

In real life, the motion of an entity can be represented by a continuous function, however, in DISSECT-CF-Fog the discrete events reflect the state of the function describing a motion, thus continuous movements are transformed into such events, for instance modifying the direction in discrete moments. Therefore, the actual position only matters and is evaluated before the decisions are made by a computing appliance or a device, for instance when the sensed data is ready to be forwarded. The mobile device movements are based on certain strategies. Currently, two mobility strategies are implemented. We decided to implement one entity and one group mobility model according to [12], but since we provide a mobility interface, the collection of usable mobility models can be easily extended.

The goal of the (i) *Nomadic* mobility model is that entities move together from one location to another, in our realisation multiple locations (i.e. targets) are available. It is very similar to the public transport of a city, where the route can be described

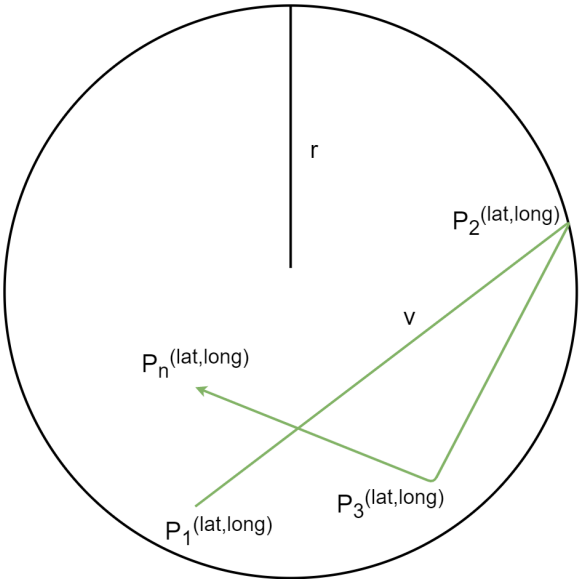


Figure 4.5: *Random Walk mobility model*

by predefined points (or bus stops), and the dedicated points (P_i) are defined as geographical positions. An entity reaching the final point of the route will no longer move but may function afterwards. Between the locations, a constant v speed is considered, and there is a fixed order of the stops as follows:

$$P_1^{(lat,long)} \xrightarrow{v} P_2^{(lat,long)} \xrightarrow{v} ... \xrightarrow{v} P_n^{(lat,long)}$$

The (ii) *Random Walk* mobility takes into consideration entities with unexpected and unforeseen movements, for instance, the observed entity walks around the city, unpredictably. The aim of this policy is to avoid moving in straight lines with a constant speed during the simulation because such movements are unrealistic. In this policy, a range of the entity is fixed (r), where it can move with a random speed (v). From time to time, or if the entity reaches the border of the range, the direction and the speed of the movement dynamically change (P_i). That kind of movement is illustrated in Figure 4.5.

The simulator monitors the position of the fog nodes and IoT devices continuously and makes decisions knowing these properties. A connection of an IoT device is closed with the corresponding node in case the latency exceeds the maximum tolerable limit of the device, or the IoT device is located outside of the range of the node. When a device finds a better fog node instead of the current one, or the IoT device runs without connection to any node, it finds an appropriate one.

As we mentioned earlier, actuation and mobility are interlinked, thus we introduce five actuator events related to mobility. Position changes are done by the *Change position* event of the actuator. The connection or disconnection methods of a device

are handled by the *Disconnect from node* and the *Connect to node* events, respectively. When a more suitable node is available for a device than the already connected one, the *Change node* actuator event is called. Finally, in some cases, a node may stay without any connection options due to its position, or in cases when only overloaded or badly equipped fog nodes are located in its neighbourhood. The *Timeout* event is used to measure the unprocessed data due to these conditions, and to empty the device's local repository, if data forwarding is not possible.

4.4.3 Evaluation

We evaluated the proposed actuator and mobility extensions of the DISSECT-CF-Fog simulator with two different scenarios, which belong to the main open research challenges in the IoT field [42]. The goal of these scenarios is to present the usability and broad applicability of our proposed simulation extension. We also extended one of the scenarios with larger-scale experiments, in order to determine the limitations of DISSECT-CF-Fog (e.g. determining the possible maximum number of simulated entities).

Our first scenario is IoT-assisted logistics, where more precise location tracking of products and trucks can be realised, than with traditional methods. It can be useful for route planning (e.g. for avoiding traffic jams or reducing fuel consumption), or for better coping with different environmental conditions (e.g. for making weather-specific decisions).

Our second scenario is IoT-assisted (or smart) healthcare, where both monitoring and reporting abilities of the smart systems are heavily relied on. Sensors worn by patients continuously monitor the health state of the observed people, and in case of data spikes, it can immediately alarm the corresponding nurses or doctors.

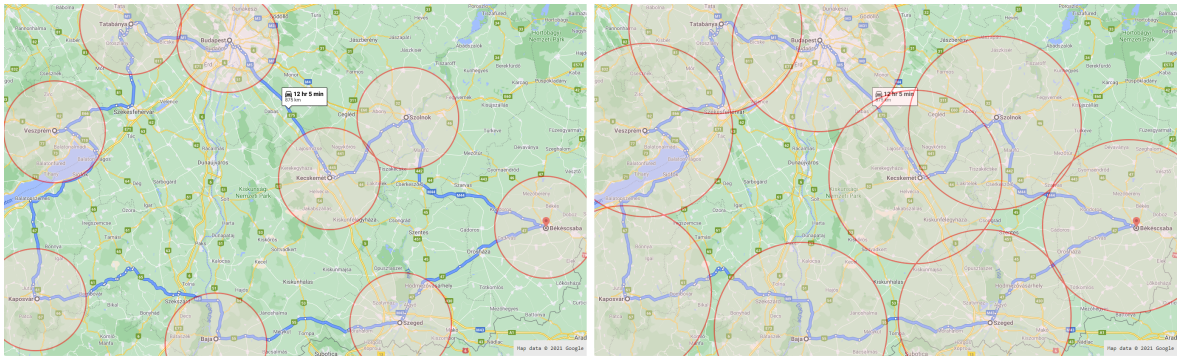
During the evaluation of our simulator extension, we envisaged a distributed computing infrastructure composed of a certain number of fog nodes (hired from local fog providers) to serve the computational needs of our IoT applications. Besides these fog resources, additional cloud resources can be hired from a public cloud provider. For each of the experiments, we used the cloud schema of LPDS Cloud of MTA SZTAKI to determine realistic CPU processing power and memory usage for the physical machines. Based on this schema we attached 24 CPU cores and 112 GB of memory for a fog node and set at most 48 CPU cores and 196 GB of memory to be hired from a cloud provider to start virtual machines (VMs) for additional data processing.

The simulator can also calculate resource usage costs, so we set VM prices according to the Amazon Web Services[19] (AWS) public cloud pricing scheme. For a cloud VM having 8 CPU cores and 16 GB RAM we set a 0.204$ hourly price (*a1.2xlarge*),

[19]Amazon Web Service (accessed in October, 2020): https://aws.amazon.com/ec2/pricing/on-demand/

while for a fog VM having 4 CPU cores and 8 GB RAM we set a 0.102$ hourly price (*a1.xlarge*). This means that the same amount of data is processed twice faster on the stronger cloud VM, however, the cloud provider also charges twice as much money for it. In our experiments, we proportionally scale the processing time of data, for every 50 KB, we model one minute of processing time on the Cloud VM.

For both scenarios, we used a PC with Intel Core i5-4460 3.2GHz, 8GB RAM, and a 64-bit Windows 10 operating system to run the simulations. Since our simulations take into account random factors, each experiment was executed ten times, and the average values are presented below.



a) 25 kilometres range b) 50 kilometres range

Figure 4.6: *Applied fog ranges in the first scenario*

The Logistics IoT Scenario

In the first scenario, we simulated a one-year-long operation of a smart transport route across cities located in Hungary. This track is exactly 875 kilometres long, and it takes slightly more than 12 hours to drive through it by a car based on Google Maps, which means the average speed of a vehicle is about 73 km/h.

We placed fog nodes in 9 different cities maintained by a domestic company, and we used a single cloud node of a cloud provider located in Frankfurt. Each fog node has a direct connection with the cloud node, the latency between them is set based on the values provided by the WonderNetwork service as before. A fog node forms a cluster with the subsequent and the previous fog node on the route as depicted in Figure 4.6. This figure also presents the first test case (a), when the range of a fog node is considered as 25 kilometres radius (similar to a LoRa network). For the second test case (b), we doubled the range to 50 kilometres radius. The IoT devices (placed in the vehicles to be monitored) were modelled with 4G network options with an average 50 ms of latency.

All vehicles were equipped with three sensors (asset tracking sensor, AIDC (automatic identification, and data capture) and RFID (radio-frequency identification))

Table 4.7: *Results of the Random actuator strategy and number of events during the first scenario*

Actuator strategy	Random					
Fog node range (km)	25			50		
Vehicle (pc.)	2	20	200	2	20	200
VM (pc.)	19	19	19	19	19	19
Generated data (MB)	48	491	4,868	79	801	8,025
Fog + Cloud cost ($)	1,988.5	2,973.1	9,619.9	3,061.1	4,026.1	10,357.4
Delay (min.)	5.0	4.01	2.03	5.0	4.02	2.02
Runtime (sec.)	3	13	141	4	16	169
Change file size (pc.)	20,937	210,009	2,102,215	34,873	348,226	3,477,983
Change node (pc.)	0	0	0	11,573	115,535	1,155,243
Change position (pc.)	181,388	1,812,784	18,137,172	181,447	1,814,159	18,141,355
Connect / disconnect to node (pc.)	12,985	129,944	1,299,099	1,556	15,833	158,751
Increase frequency (pc.)	21,239	210,352	2,104,912	34,812	346,774	3,479,261
Decrease frequency (pc.)	10,591	105,888	1,059,124	17,282	174,314	1,739,929
Restart / stop device (pc.)	0	0	0	0	0	0
Timeout (pc.)	70,941	709,384	7,091,262	0	0	0
Timeout data (MB)	27	274	2,752	0	0	0

generating 150 bytes[20] of data per sensor. A daemon service on the computational node checks the local storage for unprocessed data every five minutes and allocates them in a VM for processing. Each simulation run deals with an increasing number of IoT entities, we initialise 2, 20, and 200 vehicles in every twelve hours, which go around on the route. Half of the created objects are intended to start their movements in the opposite direction (selected randomly).

During our experiments, we considered two different actuator strategies: the (i) *Random* actuator strategy models a chaotic system behaviour, where both mobility and randomly appearing actualisation events of a sensor can happen. The failure rate of IoT components *MTTF* were set to 90% of a year, and to avoid unrealistically low or high data generation frequencies, we limited them to a range of one to 15 minutes. Finally, we enhanced the unpredictability of the system by setting the *actuation needed* to 50%. The (ii) *Transport* actuator policy defines a more realistic strategy to model asset tracking, which aims to follow objects based on a broadcasting technology (e.g. GPS). A typical use case of this is when a warehouse can prepare for receiving supplies according to the actual location of the truck. In our evaluation, if the asset was located closer than five kilometres, it would send position data every two minutes. In the case of five to 10 kilometres, the data frequency is five minutes, and from 10 to 30, the data generation is set to 10 minutes, lastly, if it is farther than 30 kilometres, it informs changes in 15 minutes.

[20]Ericsson (accessed in May, 2021): https://www.ericsson.com/en/mobility-report/articles/massive-iot-in-the-city

Table 4.8: *Results of the Transport actuator strategy and number of events during the first scenario*

Actuator strategy	Transport					
Fog node range (km)	25			50		
Vehicle (pc.)	2	20	200	2	20	200
VM (pc.)	19	19	19	19	19	19
Generated data (MB)	65	642	6,445	83	851	8,469
Fog + Cloud cost ($)	1,974.7	4,492.9	10,231.1	2,557.8	5,006.5	10,312.7
Delay (min.)	5.0	4.03	2.02	5.0	4.04	4.01
Runtime (sec.)	3	13	119	4	15	128
Change file size (pc.)	20,012	198,221	1,986,157	20,107	189,693	1,870,594
Change node (pc.)	0	0	0	6,111	65,424	654,135
Change position (pc.)	91,167	910,014	9,122,057	93,088	970,373	9,791,859
Connect / disconnect to node (pc.)	13,140	131,455	1,314,037	7,029	66,349	659,573
Increase frequency (pc.)	19,833	198,888	1,982,648	19,573	66,117	1,872,881
Decrease frequency (pc.)	19,735	199,759	1,983,997	19,646	189,298	1,875,489
Restart / stop device (pc.)	0	0	0	0	0	0
Timeout (pc.)	35,379	354,788	3,536,881	0	0	0
Timeout data (MB)	15	149	1,557	0	0	0

The results are shown in Table 4.7 and Table 4.8. The comparisons are based on the following parameters: (i) *VM* reflects the number of created VMs during the simulation on the cloud and fog nodes, which process the amount of generated data. As we mentioned earlier, our simulation tool is able to calculate the utilisation cost of the resources based on the predefined pricing schemes (*Fog+Cloud cost*). *Delay* reflects the timespan between the time of the last produced data and the last VM operation. *Runtime* is a metric describing how long the simulation runs on the corresponding PC. The rest of the parameters are previously known, it shows the number of the defined actuator and mobility events. Nevertheless, *Timeout data* is highlighting the amount of data lost, which could not be forwarded to any node, because the actual position of a vehicle is too far for all available nodes.

Interpreting the results, we can observe that in the case of the 25-kilometre range, the *Random* actuator strategy drops more than half (around 56,19%) of the unprocessed data losing information, whilst the same average is about 23,4% for the *Transport* actuator strategy. In the case of the 50-kilometre range, there is no data dropped, because the nodes roughly cover the route and the size of gaps cannot trigger the *Timeout* event. In contrast, the ranges do not cover each other in the case of the 25-kilometre range, which results in a zero *Change node* event.

Based on the *Fog+Cloud cost* metric, one can observe that the *Transport* actuator strategy utilises the cloud and fog resources more than the *Random* actuator strategy, nevertheless, the average price of a device (applying two vehicles) is about 1,197.7$.

In the case of 20 assets, it decreases to about 206.2$, and lastly, operating 200 objects reduces the price to about 50.6$, which means that the continuous load of the vehicles utilises the VMs more effectively.

Since the IoT application frequency was set to five minutes, we considered the *Delay* acceptable, when it was equal to or less than five minutes. Based on the results, all test cases fulfilled our expectations. It is worth mentioning that *MTTF* might be effective only in simulating years of operation, thus neither software nor hardware error is triggered (*Restart / stop device*) in this case. The *Runtime* metric also points to the usability and reliability performance of DISSECT-CF-Fog; less than three minutes was required to evaluate a one-year-long scenario with thousand of entities (i.e. simulated IoT devices and sensors running for a year).

Smart Healthcare Scenario

In the second scenario, we continued our experiments with a smart healthcare case study. In this scenario, patients wear blood pressure and heart rate monitors. We automatically adjust the data sampling period if the monitors report nominal behaviour: (i) in case of blood pressure lower than 90 or higher than 140; (ii) in case of heart rate values lower than 60, and higher than 100.

In this scenario, each patient represents a different data flow (starting from its IoT device), similar to the previously mentioned way. First, the data is forwarded to the fog layer, if the data processing is impossible there due to overloaded resources, then the data is moved to the cloud layer to be allocated to a VM for processing. As IoT healthcare requires as low latency as possible, the frequency of the daemon services on the computational node was set to one minute. Similar to the first scenario, one measurement of a sensor creates (a message of) 150 bytes.

We focus on the maximum number of IoT devices that can be served with minimal latency by the available fog nodes, and we are also interested in the maximum tolerable delay if the raw data is processed in the cloud. We applied the same VM parameters as in the previous scenario, and the simulation period took one day. We did not implement mobility in this scenario, nevertheless, actualisation events were still required in case of a health emergency to see how the system adapts to the unforeseen data.

Similar to the first scenario, the hospital was assumed to use a public cloud node in Frankfurt, but it was also assumed to maintain three fog nodes on the premises of the hospital. During our experiments, we considered various numbers of patients (100, 1,000, and 10,000), and we investigated how the operating costs and delay change and adapt to the difference in the number of fog VMs and actualisation events.

Since each fog node is available in the local region, the communication latency

was set randomly between 10 and 20 ms (regarding AWS[21]), furthermore, the *actuation needed* was set to 100%, because of the vital information of the sensed data, thus each measurement required some kind of actuation. The rest of the parameters were the same as we used in the logistics scenario.

Table 4.9: *Results and number of events during the second scenario*

Actuator strategy	Healthcare											
Fog / cloud node ratio	3 / 1			2 / 1			1 / 1			0 / 1		
Patient (pc.)	10,000	1,000	100	10,000	1,000	100	10,000	1,000	100	10,000	1,000	100
VM (pc.)	21	11	12	17	8	9	12	5	5	6	2	2
Generated data (MB)	251	27	2	231	27	2	197	27	2	145	27	2
Fog + Cloud cost ($)	48.1	25.1	27.8	42.0	19.7	22.7	35.2	15.3	14.9	37.1	10.8	9.9
Delay (min.)	7.74	1.41	1.06	9.51	1.46	1.07	9.50	1.79	1.05	14.8	2.44	1.22
Runtime (sec.)	8	1	1	8	1	1	8	1	1	11	1	1
Increase frequency (pc.)	132,125	14,687	1,431	119,954	14,192	1,392	98,975	14,127	1,468	71,153	13,927	1,399
Decrease frequency (pc.)	750,751	80,718	8,068	684,829	80,845	8,104	563,198	80,295	8,023	406,155	81,105	8,115
Restart / stop device (pc.)	0	0	0	0	0	0	0	0	0	0	0	0

Our findings are depicted in Table 4.9. One can observe that the increasing number of applied fog nodes reduces the average costs per patient, in the case of three fog nodes the mean cost (projected on one patient) is around 83.7$. This amount of money grows continuously as the fog nodes are omitted one by one, the corresponding average operating costs are about 97.7$, 118.7$, and 124.0$, respectively, which means maintaining fog nodes also might be economically worthwhile.

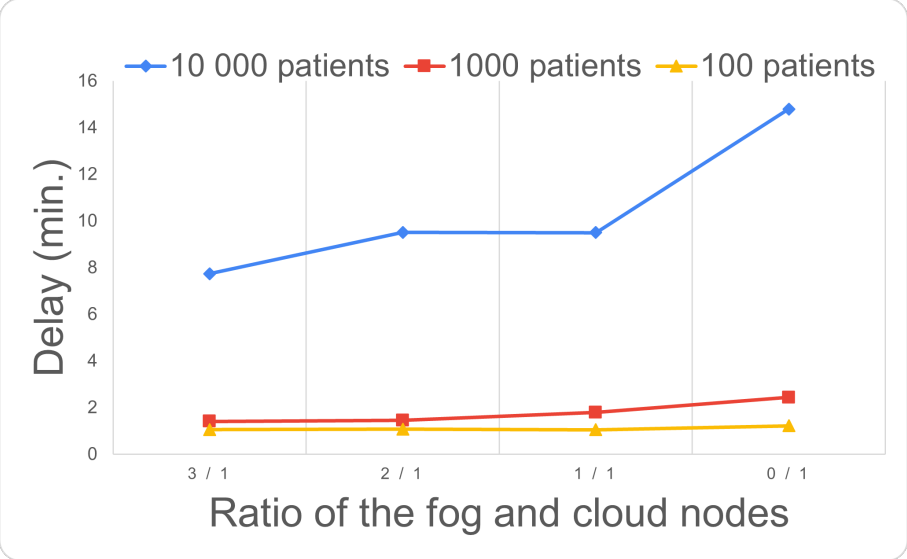


Figure 4.7: *Delay values of the second scenario*

[21]AWS Architecture Guidelines and Decisions (accessed in May, 2021): https://aws.amazon.com/blogs/compute/low-latency-computing-with-aws-local-zones-part-1/

Figure 4.7 presents the delay of the IoT application concerning the number of utilised fog and cloud nodes. Using a higher number of fog nodes can foster faster data processing, however, in the case of 10,000 patients, the best delay is 7.74 minutes, which points out that the utilised resources were overloaded. In the other cases, the system managed the patients' data with less than three minutes delay, but decreasing the number of usable fog nodes can continuously increase the delay.

Lastly, we can observe that no failure happened during the evaluation (*Restart / stop device*), because of the reliability of medical sensors and the short time of the simulation. We can also realise that our simulation tool is able to model thousands of smart objects (e.g. IoT devices and sensors), and their one-day-long simulated operation could be done in 11 seconds of elapsed time (*Runtime*) in the worst case.

Table 4.10: *Results and number of events in the scalability studies*

Actuator strategy	Healthcare				
Fog / cloud node ratio	3 / 1			7 / 1	55 / 1
Patient (pc.)	170,000	180,000	190,000	190,000	190,000
VM (pc.)	24	24	Out of memory	48	336
Generated data (MB)	1,196	1,261		1,513	1,679
Fog + cloud cost ($)	197.6	208.5		244.9	674.9
Delay (min.)	6,256.0	6,796.0		5,886.0	9.9
Runtime (sec.)	186	256		159	163
Increase frequency (pc.)	624,860	657,725		790,999	810,153
Decrease frequency (pc.)	3,557,783	3,751,754		4,498,906	4,049,325
Restart / stop device (pc.)	0	0	0	0	0

Large-scale Experiments of the Smart Healthcare Scenario

In this section, our goal was to point out the possible limitations of DISSECT-CF-Fog using the previously detailed smart healthcare scenario. The runtime of DISSECT-CF-Fog largely depends on the used execution environment and its actual hardware resources (mostly memory), similar to any other software.

Our findings are presented in Table 4.10, in which we used the same metrics as before.

For this scalability study, we also applied the earlier used topology with three fog nodes and a cloud node. Determining the exact number of IoT devices that can be modelled by the simulator is not possible because our system takes into account random factors. Nevertheless, we can give an estimate by scaling the number of IoT devices, in our case the number of active devices (i.e. patients).

In this evaluation, we increased the number of patients with 10,000 for the test cases, and examined the memory usage of the execution environment. The results showed that even for cases of 170,000 and 180,000 IoT devices, the fog and cloud

nodes can process the vast amount of data generated by the modelled IoT sensors, however, the *Delay* value also increased dramatically to 6,256 minutes, in the first case, and 6,796 minutes, in the second case. It is worth mentioning that besides such a huge number of active entities, the *Runtime* values are below five minutes. When we simulated 190,000 IoT devices, the simulator consumed all of the memory of the underlying hardware.

In the fourth test case, we applied seven fog nodes. Our findings showed that the *Delay* value decreased spectacularly to 5,886 minutes, however, it is far from what we experienced in the second scenario, therefore, our further goal was to define how many computational resources (i.e. fog nodes) are required to decrease the *Delay* parameter below ten minutes, similar to what we expected in the second scenario.

We can clearly see in the fifth test case that at least 55 fog nodes are required for 190,000 IoT devices to process and store their data. In this case, the *Delay* value is 9.9 minutes, but because of the higher number of computational nodes, both numbers of the utilised VMs (336 pieces) and these costs (674.9$) increased heavily. The Java representation of the fog and cloud nodes hardly differ, therefore, we could reach similar results if we increased the number of cloud nodes as well.

It can be clearly seen that the critical part of DISSECT-CF-Fog is the number of IoT devices utilising in the system, however, if we also increase the number of the simulated computing resources (i.e. fog and cloud nodes), we can reach better scalability (i.e. the delay and simulation runtime would not grow). The reason for this is that the actual Java implementation of DISSECT-CF-Fog stores the references of model entities of the devices and the unprocessed data. To conclude, the current DISSECT-CF-Fog extension is capable of simulating even up to 200 thousand system entities. Limitations are only imposed by the hardware parameters utilised, and the wrongly (or extremely) chosen ratio of the number of IoT devices and computing nodes set for the experiments.

4.5 Modelling Energy Consumption in DISSECT-CF-Fog

The monitoring and measuring of energy consumption entail significant challenges for IoT-Fog-Cloud systems since the task of offloading and resource allocation of an IoT-Fog-Cloud architecture can take it into consideration. The initial energy model of DISSECT-CF covered cloud data centres by introducing resource consumption modelling for CPU, disk, and network energy utilisation. To ensure the required level of system granularity, the simulator mimics the behaviour of infrastructure clouds by predefined states of physical machines (PM), virtual machines (VM), storage, and disks. For instance, a PM can be in the following states: turned off, switching on, running, and switching off. As a result, the basic concept of energy saving can be easily realised by turning off the unused machine. Besides, this refined model sup-

ports the mapping of certain energy consumption values to the predefined states, which ensures the fine-granularity of the simulator.

The energy model of the simulator takes into account: the minimum (min) power (e.g. the machine/device is turned off, but still plugged into the energy source), the maximum (max) power (e.g. if the CPU is fully utilised), and the idle power (e.g. when the PM is running without executing computational tasks). At this moment the simulator has two power models: (i) dynamic power draining behaviour applies linear interpolation between idle and max power values, whilst (ii) constant power draining behaviour can consider any power value (e.g. min). By default, the dynamic model is applied in the case of states with high energy consumption (e.g. running state of a PM), and it handles the idle power with min power values, and the consumption range can be get by subtracting the idle power value from the max power value. In this section, we take a step forward, and besides the energy measurement of cloud resources, we cover IoT devices with our proposed extended model, to enable complex energy utilisation analysis of IoT-Fog-Cloud systems. First, we started to analyse the real power consumption of microcontrollers, which is detailed in the next subsection.

Table 4.11: *Uniform sampling of microcontrollers*

Microcontroller	ESP32			Raspberry Pi (RPi)		
Sampling (min.)	V	I	P	V	I	P
1	5.13	0.07	0.3591	5.12	0.28	1.4336
2	5.17	0.02	0.1034	5.12	0.26	1.3312
3	5.19	0.02	0.1038	5.12	0.31	1.5872
4	5.17	0.02	0.1034	5.13	0.26	1.3338
5	5.19	0.02	0.1038	5.13	0.28	1.4364
6	5.17	0.02	0.1034	5.12	0.28	1.4336
7	5.16	0.05	0.258	5.13	0.28	1.4364
8	5.19	0.02	0.1038	5.13	0.26	1.3338
9	5.21	0.02	0.1042	5.12	0.28	1.4336
10	5.17	0.02	0.1034	5.13	0.28	1.4364
11	5.18	0.02	0.1036	5.13	0.31	1.5903
12	5.17	0.02	0.1034	5.13	0.28	1.4364
13	5.20	0.05	0.26	5.12	0.28	1.4336
14	5.21	0.02	0.1042	5.13	0.28	1.4364
15	5.18	0.02	0.1036	5.13	0.26	1.3338

Table 4.12: *Mapping the benchmark and measured values to the model power values in DISSECT-CF-Fog*

Data source		Research papers		Websites		Our experiments		IoT energy model	
Microcontroller		ESP32	RPi	ESP32	RPi	ESP32	RPi	ESP32	RPi
Power cons. (W)	min	N.A.	N.A.	0.01	0.1	N.A.	N.A.	0.01	0.1
	idle	0.17	0.94	0.04	1.1	0.1	1.33	0.1	1.1
	max	0.28	1.57	0.42	2.1	0.36	1.59	0.35	1.75

4.5.1 Analysis of Real Microcontrollers

In order to determine a fine-grained energy model for microcontrollers, we measured and collected the energy consumption values of real devices. In our experiments, we chose ESP32[22] (WROOM-32) and Raspberry Pi[23] (1 Model A+) microcontrollers for further analysis. Both devices were equipped with DTH22 temperature and humidity sensors, and a KCX-017 meter was applied to display the voltage and the current of the connected USB port. The assembly of the used gadgets can be seen in Figure 4.8.

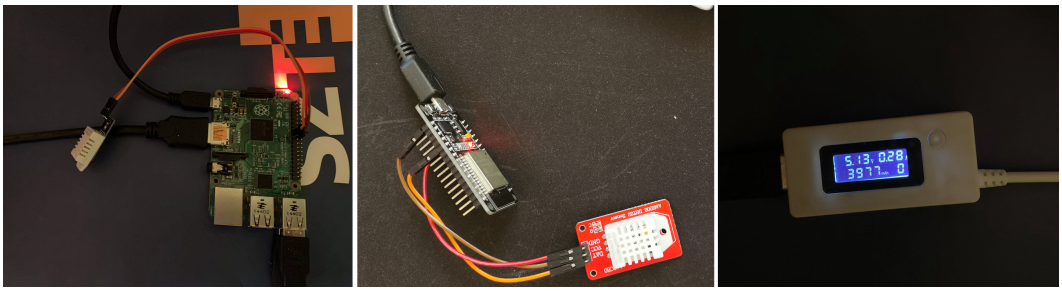


Figure 4.8: *The utilisation of Raspberry Pi (left) and ESP32 (middle) microcontrollers and KCX-017 meter (right)*

To measure the general power consumption of IoT applications, we developed a typical and simple program written in Python/MicroPython covering the following functionalities: sensor data reading (temperature and humidity values in our current case), message creation, and sending as an IoT client device by using the MQTT protocol. We scheduled sensor value sampling every minute by default and connected the devices to the Internet via WiFi. The data application running on the microcontrollers forwarded the sensor data to an IoT analytics platform called Thingspeak[24], where it could be visualised.

To determine the electric power (P measured in watts) in the SI system, we mul-

[22]The official website of ESP32 (accessed in February, 2021): https://www.espressif.com/en/products/socs/esp32/

[23]The official website of Raspberry Pi (accessed in February, 2021): https://www.raspberrypi.org/

[24]The official website of Thingspeak (accessed in February, 2021): https://thingspeak.com/

tiplied the metered voltage (V measured in volts) with the metered electric current (I measured in amps) values:

$$P = V * I, e.g. 1W = 1V * 1A \tag{4.1}$$

Finally, we can determine the energy usage (J measured in joules/watt-second/kilowatt-hour) by:

$$J = P * t, e.g. 1J = 1W * 1s \tag{4.2}$$

4.5.2 The Energy Model for IoT Devices

Finally, in order to utilize monitored data of real IoT devices in DISSECT-CF-Fog, we executed our sampling application five times on both microcontrollers for 15 minutes, while measuring the power consumption each millisecond.

Table 4.11 presents the average values of the uniform sampling of the metering device for each one-minute period. Based on the monitored values, we calculated the electric power. The results show that our typical IoT monitoring application consumed 0.1 to 0.36 W per minute on average with ESP32 and 1.3 to 1.59 W with RPi. In the next subsection, we show how we applied these measured values to our proposed IoT energy model.

Concerning the power consumption of IoT resources, we had to build up the energy model from scratch. In this work, we had to extend the IoT device representation of DISSECT-CF-Fog, which represents any smart objects, and is responsible for power consumption metering for IoT devices during simulations. To resolve this issue, we decided to create a more detailed physical layer called a microcontroller for implementing our energy model. Such realisation keeps the already existing functionalities (e.g. data sensing of IoT sensors, temporary data storing, and data forwarding to fog or cloud nodes), and introduces predefined states for microcontrollers, which allow mapping a certain power consumption to a certain state.

Besides our real measurements of a typical use of a microcontroller, we gathered information from the following works. [40] and [31] focus on the comparative analysis and the monitoring of ESP32 and Raspberry Pi devices, while detailed online benchmark results for their energy consumption can also be found on websites[25] [26]. The collected and measured numbers are shown and compared in Table 4.12. It also shows the predefined values (for min, idle, and max) we chose to be the base for our IoT energy model. We arrived at these values by counting the median for the concrete

[25]Raspberry Pi benchmark values (accessed in February, 2021): https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/

[26]ESP32 benckmark values (accessed in February, 2021): https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/

values gathered from the research papers, websites and by our measurements.

Our findings and experiments revealed that the power consumption values of microcontrollers are highly dependent on their actual behaviour and their use cases. Typical modifying circumstances may be the usage of a wired connection instead of wireless, and/or different types of power supply cables or converters. As we mentioned earlier, during our experiments we used an online service for retrieving and storing the generated data by the DTH22 sensor. Table 4.11 also shows that in a few cases (i.e. after every fifth sampling), the consumption doubled in the case of ESP32. To handle such extreme cases and to be able to simulate uncertainty, we introduce three different states of a microcontroller in our model.

The state *OFF* indicates a fully turned-off device with static minimal energy consumption using the min power preset value. The *RUNNING* state represents a high energy consumption state, where the actual power consumption can change dynamically with regard to the actual CPU utilisation. The minimal and maximal consumption values in this state are set by the predefined idle and max power values. To simulate specific events when high power spikes appear (caused by e.g. activating a previously unused port of the device), we introduce the *ACTIVE* state. It also represents a high energy consumption state allowing dynamic changes, but its minimal value should be higher than in the *RUNNING* state; by default, it is set to double the idle power value.

According to our observation, we experienced such behaviour in 20% of the sampling process, therefore, we decided that the *ACTIVE* state will be set during the sensing process of IoT sensors until the data is saved into the local storage of the IoT device. In this way, each simulated IoT device enters the *OFF* state when it is created, the *RUNNING* state, when it is started, and it periodically switches between the *ACTIVE* (performing sensor data generation) and *RUNNING* states till it is stopped (*OFF* state) or terminated.

We would also note that DISSECT-CF-Fog provides a transparent and easily usable interface to create additional, new states, and hence multiple energy models, and it is up to the researcher where to use such new states during the simulation.

4.5.3 Evaluation of the Energy Extension

In this section, we illustrate the use of the extended, unified energy model for IoT-Fog-Cloud architectures in DISSECT-CF-Fog. For this purpose, we model one of the typical IoT use cases, which represents a weather forecasting scenario with numerous weather stations (run by IoT microcontrollers with special sensors). These devices can communicate with a fog layer directly, which contains three different nodes with an equal amount of resources, utilising 40 CPU cores and 40 GB of memory in total. On the top of the fog topology, there is one cloud data centre having 56 CPU

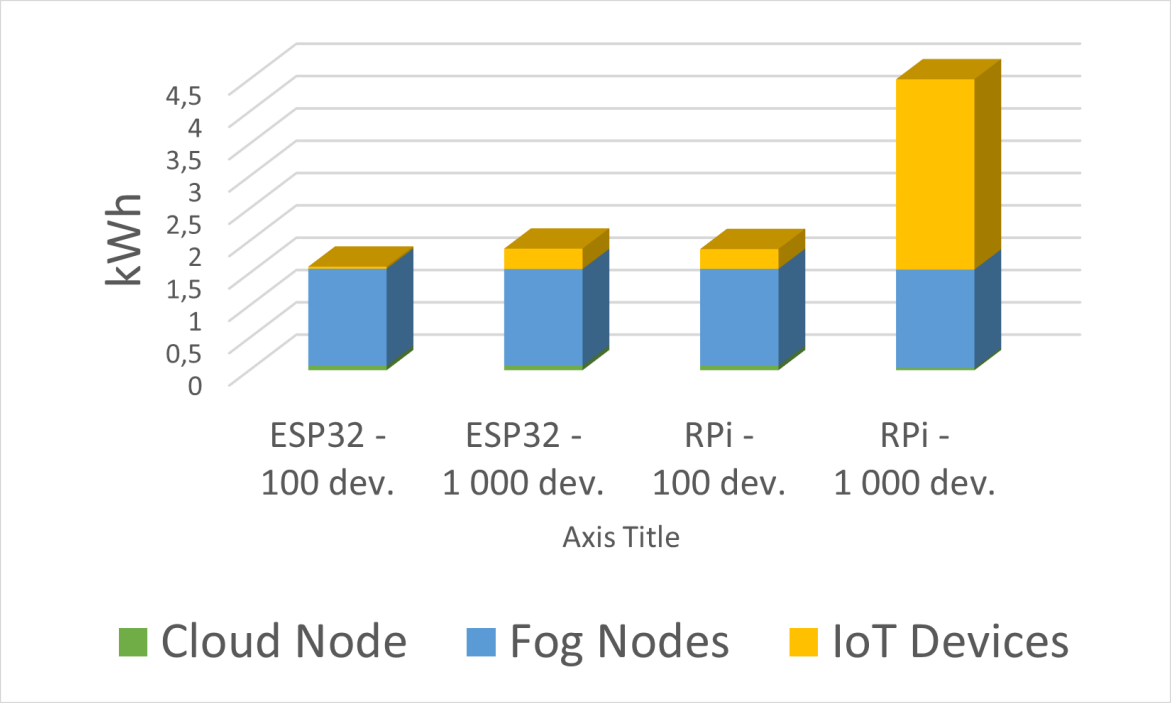


Figure 4.9: *Cumulative energy consumption of cloud, fog nodes and IoT devices*

cores and 40 GB of memory, furthermore, the devices are not allowed to send messages (unprocessed sensor data) directly to the cloud (they are connected only to the fog). We considered two types of virtual machine images simulating existing Amazon Cloud (AWS) instances. The *cloud1* node can utilise VMs with 8 CPU cores and 4 GB of memory, their hourly prices were set to 0.202$, while the *fog1*, *fog2* and *fog3* nodes can deploy VMs with 4 CPU cores and 2 GB of memory with 0.101$ hourly price. We also set the IoT-side pricing by applying the IBM Cloud pricing schema, which charges the consumer after the amount of data exchanged (in MB).

In our simulation, the microcontrollers can use either ESP32 or Raspberry Pi energy models, and they are equipped with a temperature-humidity sensor (similar to our real-world measurements). In our weather forecasting use case, we defined three different scenarios by scaling up the number of operating devices. In the first case, we utilised 100 IoT devices, then we increased the number of devices to 1,000, and finally, in the last case, the maximum device number was 10,000, operated for 60 minutes within the experiments. The microcontrollers measured the environmental parameters every 60 seconds, similar to the real device evaluation, hence our goal was to map the real monitoring execution in the DISSECT-CF-Fog simulation environment.

The evaluation process is the following in each scenario: (i) the IoT microcontrollers monitor the environment based on their sampling frequency, (ii) the gener-

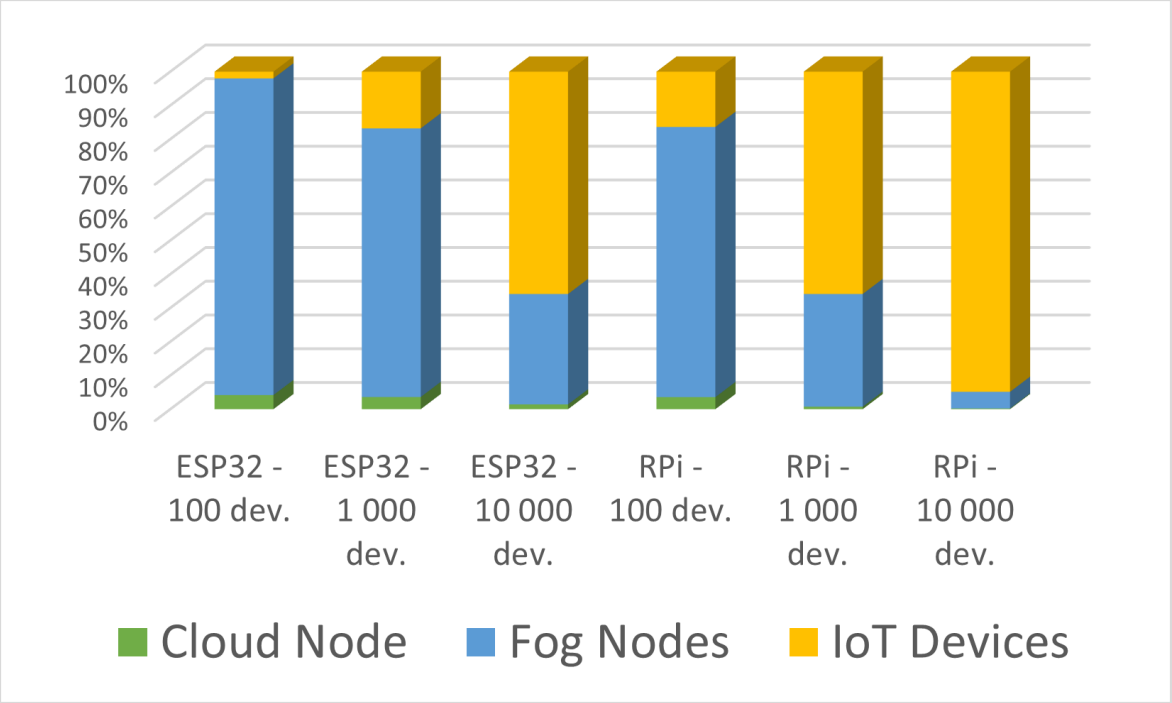


Figure 4.10: *Energy consumption percentage of cloud, fog nodes and IoT devices*

ated data are forwarded to the less loaded fog node (using the default scheduling algorithm), (iii) a node allocates a task (i.e. collection of 256 KB of data) to a VM to be processed, or requests a new one, if there is no free VM available, in case the current resource capacity allows it. Otherwise, the unallocated task will be moved to a less-loaded node (in the fog or to the cloud layer).

During the evaluation, we modelled a European-wide scenario, where the cloud was located in Frankfurt, whilst the three fog nodes were positioned in London, Budapest, and Vienna. The latency between them was determined based on online ping statistics[27]. The delay between a device and a fog node was set to an average of 50 ms weighted with the actual physical distance, and the positions of the devices were randomly generated across Europe.

In order to highlight the energy consumption of the nodes, the number of VMs was scaled up and down dynamically according to the actual load caused by the tasks. To be as realistic as we can, each computational resource dealt with different energy models based on the resource schema of LPDS Cloud of MTA SZTAKI. The exact values we used to set the energy model parameters are summarised in Table 4.12 and Table 4.14.

The comparison of the results can be seen in Table 4.13. We listed the number of VMs utilised by all nodes (*Number of VMs*), and the cost of both the cloud/fog and IoT

[27]WonderNetwork (accessed in March, 2021): https://wondernetwork.com/pings/

Table 4.13: *Comparison of the final results of the simulated scenarios*

Microcontroller		ESP32			Raspberry Pi		
Number of devices		100	1,000	10,000	100	1,000	10,000
Number of VMs		7	7	29	7	7	29
Cloud and fog cost ($)		0.84	0.85	2.79	0.84	0.84	2.79
IoT cost ($)		0.009	0.90	83.2	0.009	0.90	83.2
Delay (min.)		1.03	1.25	3.50	1.03	1.25	3.50
Runtime (sec.)		0	4	95	0	4	118
Energy consumption (kWh)	*cloud1* consumption	0.067	0.068	0.068	0.067	0.034	0.068
	fog1 consumption	0.456	0.456	0.456	0.456	0.456	0.456
	fog1 device consumption	0.006	0.053	0.525	0.053	0.500	5.000
	fog2 consumption	0.436	0.431	0.531	0.433	0.456	0.492
	fog2 device consumption	0.011	0.106	1.050	0.105	1.003	10.000
	fog3 consumption	0.611	0.611	0.578	0.611	0.611	0.611
	fog3 device consumption	0.016	0.158	1.575	0.150	1.500	14.972
Total consumption by nodes		1.570	1.569	1.636	1.569	1.558	1.628
Total consumption by devices		0.032	0.316	3.150	0.307	3.003	29.972

Table 4.14: *The chosen values of the energy model for nodes and microcontrollers*

Resource Type	Min Power	Idle Power	Max Power
cloud1	20	398	533
fog1	20	296	493
fog2	20	296	533
fog3	20	398	493

sides (*Cloud and fog cost, IoT cost*). The *Delay* value reflects the makespan of the IoT application, whilst *Runtime* indicates the elapsed time in the execution environment required by the actual simulation. The *Energy consumption* ensures consumption information detailed for each computational node (e.g. *cloud1* consumption denotes the consumed energy by the *cloud1* node), and we also counted the summed consumption values of IoT devices related to an actual node (e.g. *fog1 device consumption* denotes the total consumed energy by all simulated microcontrollers connected to *fog1*). Lastly, the total energy usage of both nodes and devices is presented by *Total consumption of nodes* and *Total consumption of devices*.

As we can see from Table 4.13, the cloud resource utilisation is basically the same in all six simulation cases, because they had to deal with around the same amount of unprocessed data/tasks (coming from the fog layer). Nevertheless, it also shows that

in the case of 1,000 devices, seven VMs could easily handle the scheduled amount of tasks for both microcontrollers. The more data a task contains, the more time it takes for the task to be processed, and additional incoming tasks may trigger new VMs to be deployed (depending on the applied task scheduling policy threshold).

In the third case having 10,000 devices, the number of VMs is dramatically increased to 29, for both device types.

Since the IBM Cloud pricing is independent of the actual device type, only the transmitted data counts and the cost of the computational nodes is proportional to the number of utilised VMs, therefore, the corresponding costs are the same in the case of ESP32 and RPi. It can also be observed that the timeout delay (i.e. application makespan minus the set operation interval of the IoT devices (60 minutes in these scenarios)) is less than 90 seconds for 100 and 10,000 devices. As we can see for the third, 10,000 devices cases, the throughput of the system decreased, hence the delay increased to 3.5 minutes. The execution time (*runtime*) of the simulations for all cases remains within two minutes for all cases, which points out that DISSECT-CF-Fog can manage thousands of entities on a single PC (for the evaluation we used a PC with Windows 10 OS, i5-4460 CPU, and 8 GB memory).

Figure 4.9 and Figure 4.10 highlight the results by comparing the energy consumption ratio of the utilised cloud node, fog nodes, and IoT devices (i.e. microcontrollers). Figure 4.9 depicts the total energy used in kWh for each category, while Figure 4.10 depicts their ratio in percentage. As we can see from the diagrams, cloud consumption takes only a small part of the total energy consumption in all six scenarios. The fog nodes are mostly capable of handling the vast amount of data with their own resources generated by the IoT layer, and there is no need to involve cloud resources drastically. Nevertheless, when we scale up the number of microcontrollers in the IoT layer, our results show a significant increase in the total energy consumption, caused by only exclusively the operation of the IoT devices. For the case of using 10,000 RPi devices, we can see that the energy consumed by the IoT layer takes up almost 95% of the total consumption, as shown in Figure 4.10. For smaller scales, we can observe that 100 ESP32 devices caused only 2% of the total energy consumption. This ratio goes up to about 16%, in the case of 1,000 ESP32 and 100 RPi devices, and we experienced around the same ratio in the case of 10,000 ESP32 and 1,000 RPi devices (with $\sim$ 66%).

4.6 Discussion and Concluding Remarks

In the last chapter of this thesis, we presented three extensions to the DISSECT-CF-Fog simulator. To evaluate the extensions, we used the basic weather forecasting use case and some more complex case studies of frequently used IoT applications. First, we extended the simulation environment with a more detailed fog model to enhance

its location awareness and multi-layer fog node management by introducing various application strategies to manage task offloading decisions of complex IoT-Fog-Cloud systems.

Next, we introduced the extended version of DISSECT-CF-Fog supporting actuators and mobility features. Concerning our main, novel contribution in this chapter, we designed and developed an actuator model that enables broad configuration possibilities for investigating IoT-Fog-Cloud systems. With our extensions, various IoT device behaviours and management policies can be defined and evaluated with ease in this simulator. We also presented how to use different actuator strategies, in order to define specific application (and sensor/actuator) behaviour.

Finally, we presented a novel extension of the energy model of the DISSECT-CF-Fog simulator to enable the energy monitoring of its simulated IoT components. In this way, we realised a unified energy model capable of analysing the power consumption of complex, IoT-Fog-Cloud infrastructures.

The results of this chapter belong to **Thesis III**, and its contents were published in papers [P4], [P8], and [P9]. My contributions presented in this chapter are the following:

III/1. I designed a generic model of Fog Computing and implemented it in the DISSECT-CF-Fog simulator to enable the modelling of the Cloud-to-Thing Continuum.

III/2. I proposed greedy and Pliant-based task allocation algorithms for fog and cloud infrastructure management to optimise IoT application makespan, utilisation costs, and energy consumption.

III/3. I designed a realistic and dynamic IoT behaviour modelling supporting IoT mobility features, which can be configured by using the novel actuator interface of DISSECT-CF-Fog.

III/4. I designed an IoT energy model based on real-world experiments and proposed an extension of the DISSECT-CF-Fog simulator for the energy usage monitoring of IoT devices.

III/5. I also presented an enhancement of the basic weather forecasting IoT scenarios, towards modelling more refined IoT logistics and healthcare case studies.

Bibliography

[1] N. Abbas, M. Asim, N. Tariq, T. Baker, and S. Abbas. A mechanism for securing iot-enabled applications at the fog layer. *Journal of Sensor and Actuator Networks*, 8:1, 2019.

[2] D. Perez Abreu, K. Velasquez, M. Curado, and E. Monteiro. A comparative analysis of simulators for the cloud to fog continuum. *Simulation Modelling Practice and Theory*, 101:102029, 2020.

[3] V. Angelakis, I. Avgouleas, N. Pappas, and D. Yuan. Flexible allocation of heterogeneous resources to services on an iot device. *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 99–100, 2015.

[4] H. F. Atlam, R. J. Walters, and G. B. Wills. Fog computing and the internet of things: A review. *Big Data and Cognitive Computing*, 2:2, 2018.

[5] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy. Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study. *IEEE Access*, pages 9882–9910, 2017.

[6] M. Bendechache, S. Svorobej, P. Takako Endo, and T. Lynn. Simulating resource management across the cloud-to-thing continuum: A survey and future directions. *Future Internet*, 12:95, 2020.

[7] A. Botta, W. de Donato, V. Persico, and A. Pescapé. Integration of cloud computing and internet of things: A survey. *Future Generation Computer Systems*, 56:684–700, 2016.

[8] A. Brogi, S. Forti, and A. Ibrahim. Deploying fog applications: How much does it cost, by the way? *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER)*, pages 68–77, 2018.

[9] M. Bux and U. Leser. Dynamiccloudsim: Simulating heterogeneity in computational clouds. *Future Generation Computer Systems*, 46:85–99, 2015.

[10] R. N. Calheiros, M. A. S. Netto, C. De Rose, and R. Buyya. Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. *Software: Practice and Experience*, 43:595–612, 2012.

[11] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41:1, 2011.

[12] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, 2:10.1002/wcm.72, 2002.

[13] D. Chen, D. Irwin, and P. Shenoy. Smartsim: A device-accurate smart home simulator for energy analytics. *IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 686–692, 2016.

[14] B. Costa, P. F. Pires, and F. C. Delicato. Modeling iot applications with sysml4iot. *42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 157–164, 2016.

[15] A. V. Dastjerdi and R. Buyya. Fog computing: Helping the internet of things realize its potential. *Computer*, 49:112–116, 2016.

[16] J. Dombi. A general class of fuzzy operators, the demorgan class of fuzzy operators and fuzziness measures induced by fuzzy operators. *Fuzzy sets and systems*, 8:149–163, 1982.

[17] J. Dombi. Pliant system. *In Proceedings of IEEE International Conference on Intelligent Engineering Systems*, pages 289–294, 1997.

[18] S. Frey F. Fittkau and W. Hasselbring. Cdosim: Simulating cloud deployment options for software migration support. *6th IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, pages 37–46, 2012.

[19] C. Fiandrino, A. Capponi, G. Cacciatore, D. Kliazovich, U. Sorger, P. Bouvry, B. Kantarci, F. Granelli, and S. iordano. Crowdsensim: a simulation platform for mobile crowdsensing in realistic urban environments. *IEEE Access*, 5:3490–3503, 2017.

[20] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire. Cloudsim plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 400–406, 2017.

[21] S. K. Garg and R. Buyya. Networkcloudsim: Modelling parallel applications in cloud simulations. *Fourth IEEE International Conference on Utility and Cloud Computing (UCC)*, pages 105–113, 2011.

[22] Z. Ghanbari, N. Jafari Navimipour, M. Hosseinzadeh, and A. Darwesh. Resource allocation mechanisms and approaches on the internet of things. *Cluster Computing*, 22:1253–1282, 2019.

[23] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47:1275–1296, 2017.

[24] S. N. Han, G. M. Lee, N. Crespi, K. Heo, N. Van Luong, M. Brut, and P. Gatellier. Dpwsim: A simulation toolkit for iot applications using devices profile for web services. *IEEE World Forum on Internet of Things (WF-IoT)*, pages 544–547, 2014.

[25] W. A. Higashino, M. A.M. Capretz, and L. F. Bittencourt. Cepsim: Modelling and simulation of complex event processing systems in cloud environments. *Future Generation Computer Systems*, 65:122–139, 2016.

[26] C. Hong and B. Varghese. Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. *ACM Computing Surveys*, 52:5, 2019.

[27] F. Howell and R. McNab. Simjava: A discrete event simulation library for java. *Simulation Series*, 30:51–56, 1998.

[28] M. Héder, E. Rigó, D. Medgyesi, R. Lovas, Sz. Tenczer, F. Török, A. Farkas, M. Emődi, J. Kadlecsik, Gy. Mező, Á. Pintér, and P. Kacsuk. The past, present and future of the elkh cloud. *Információs Társadalom*, 22:128, 2022.

[29] Y. Jararweh, Z. Alshara, M. Jarrah, M. Kharbutli, and M. N. Alsaleh. Teachcloud: A cloud computing educational toolkit. *International Journal of Cloud Computing*, 2:237–257, 2013.

[30] D. N. Jha, K. Alwasel, A. Alshoshan, X. Huang, R. Naha, S. Battula, S. Garg, D. Puthal, P. James, A. Zomaya, S. Dustdar, and R. Ranjan. Iotsim-edge: A simulation framework for modeling the behavior of internet of things and edge computing environments. *Software: Practice and Experience*, 50:844–867, 2020.

[31] F. Kaup, P. Gottschling, and D. Hausheer. Powerpi: Measuring and modeling the power consumption of the raspberry pi. *39th Annual IEEE Conference on Local Computer Networks*, pages 236–243, 2014.

[32] G. Kecskemeti. Dissect-cf: A simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58:188–218, 2015.

[33] A. Kertesz, J. D. Dombi, and A. Benyi. A pliant-based virtual machine scheduling solution to improve the energy efficiency of iaas clouds. *Journal of Grid Computing*, 14:41–53, 2016.

[34] A. Kertesz, T. Pflanzner, and T. Gyimothy. A mobile iot device simulator for iot-fog-cloud systems. *Journal of Grid Computing*, 17:529–551, 2018.

[35] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. *IEEE Conference and Exhibition on Global Telecommunications (GLOBECOM)*, pages 1–5, 2010.

[36] I. Lera, C. Guerrero, and C. Juiz. Yafs: A simulator for iot scenarios in fog computing. *IEEE Access*, 7:91745–91758, 2019.

[37] Z. Li, K. Liu, Y. Su, and Y. Ma. Adaptive resource allocation algorithm for internet of things with bandwidth constraint. *Transactions of Tianjin University*, 18:253–258, 2012.

[38] M. M. Lopes, W. A. Higashino, M. A.M. Capretz, and L. F. Bittencourt. 10.1145/3147234.3148101. *Proceedings of The 10th International Conference on Utility and Cloud Computing*, page 47–52, 2017.

[39] R. Mahmud, R. Kotagiri, and R. Buyya. Fog computing: A taxonomy, survey and future directions. *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, pages 103–130, 2018.

[40] A. Maier, A. Sharp, and Y. Vagapov. Comparative analysis and practical implementation of the esp32 microcontroller module for the internet of things. *International Conference on Internet Technologies and Applications*, pages 143–148, 2017.

[41] Z. Mann. Cloud simulators in the implementation and evaluation of virtual machine placement algorithms. *Software: Practice and Experience*, 48:7, 2017.

[42] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob. Big iot data analytics: Architecture, opportunities, and open research challenges. *IEEE Access*, 6:5247–5261, 2017.

[43] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran. Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. *IEEE Fog World Congress (FWC)*, pages 1–6, 207.

[44] N. Mohan and J. Kangasharju. Edge-fog cloud: A distributed cloud for internet of things computations. *Cloudification of the Internet of Things (CIoT)*, pages 1–6, 2016.

[45] M. I. Naas, J. Boukhobza, P. Raipin Parvedy, and L. Lemarchand. An extension to ifogsim to enable the design of data placement strategies. *IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–8, 2018.

[46] K. Nahrstedt, H. Li, P. Nguyen, S. Chang, and L. Vu. Internet of mobile things: Mobility-driven challenges, designs and implementations. *IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 25–36, 2020.

[47] P. G. V. Naranjo, Z. Pooranian, M. Shojafar, M. Conti, and R. Buyya. Focan: A fog-supported smart city network architecture for management of applications in the internet of everything environments. *Journal of Parallel and Distributed Computing*, 132:274–283, 2018.

[48] E. C. H. Ngai, M. R. Lyu, and Jiangchuan Liu. A real-time communication framework for wireless sensor-actuator networks. *IEEE Aerospace Conference*, 9, 2006.

[49] Z. Nikdel, B. Gao, and S. W. Neville. Dockersim: Full-stack simulation of container-based software-as-a-service (saas) cloud deployments and environments. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 1–6, 2017.

[50] A. Núñez, J. L. Vázquez-Poletti, A. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente. Icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10:185–209, 2012.

[51] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer. Groudsim: An event-based simulation framework for computational grids and clouds. *Euro-Par 2010 Parallel Processing Workshops*, pages 305–313, 2011.

[52] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya. Containercloudsim: An environment for modeling and simulation of containers in cloud data centers. *Software: Practice and Experience*, 47(4):505–521, 2017.

[53] F. Pisani, F. M. C. de Oliveira, E. S. Gama, R. Immich, L. F. Bittencourt, and E. Borin. Fog computing on constrained devices: Paving the way for the future iot. *Advances in Edge Computing: Massive Parallel Processing and Applications*, 35:22–60, 2020.

[54] C. Puliafito, D. M. Gonçalves, M. M. Lopes, L. L. Martins, E. Madeira, E. Mingozzi, O. Rana, and L. F. Bittencourt. Mobfogsim: Simulation of mobility and migration for fog computing. *Simulation Modelling Practice and Theory*, 101:102062, 2020.

[55] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology*, 19:2, 2019.

[56] T. Qayyum, A. W. Malik, M. A. K. Khattak, O. Khalid, and S. U. Khan. Fognetsim++: A toolkit for modeling and simulation of distributed fog environment. *IEEE Access*, 6:63570–63583, 2018.

[57] L. Qian, Luo Z, Y. Du, and L. Guo. Cloud computing: An overview. *IEEE International Conference on Cloud Computing*, page 626–631, 2009.

[58] U. U. Rahman, K. Bilal, A. Erbad, O. Khalid, and S. U. Khan. Nutshell—simulation toolkit for modeling data center networks and cloud computing. *IEEE Access*, 7:19922–19942, 2019.

[59] P. F. Roth. Discrete, continuous, and combined simulation. *Proceedings of the 20th Conference on Winter Simulation*, pages 56–60, 1988.

[60] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov. Spanedge: Towards unifying stream processing over central and near-the-edge data centers. *EEE/ACM Symposium on Edge Computing (SEC)*, pages 168–178, 2016.

[61] M. Seufert, B. K. Kwam, F. Wamser, and P. Tran-Gia. Edgenetworkcloudsim: Placement of service chains in edge clouds using networkcloudsim. *IEEE Conference on Network Softwarization (NetSoft)*, pages 1–6, 2017.

[62] D. Miorandi S. Sicari, F.De Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10:1497–1516, 2012.

[63] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya. Cloudsimsdn: Modeling and simulation of software-defined cloud data centers. *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 475–484, 2015.

[64] C. Sonmez, A. Ozgovde, and C. Ersoy. Edgecloudsim: An environment for performance evaluation of edge computing systems. *Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 39–44, 2017.

[65] S. Sotiriadis, N. Bessis, E. Asimakopoulou, and N. Mustafee. Towards simulating the internet of things. *28th International Conference on Advanced Information Networking and Applications Workshops,*, pages 444–448, 2014.

[66] I. Sriram. Speci, a simulation tool exploring cloud-scale data centres. *Cloud Computing*, page 381–392, 2009.

[67] S. Svorobej, P. Takako Endo, M. Bendechache, C. Filelis-Papadopoulos, K. M. Giannoutakis, G. A. Gravvanis, D. Tzovaras, J. Byrne, and T. Lynn. Simulating fog and edge computing scenarios: An overview and research challenges. *Future Internet*, 11:3, 2019.

[68] R. Taylor, D. Baron, and D. Schmidt. The world in 2025 - predictions for the next ten years. *10th International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT)*, pages 192–195, 2015.

[69] D. Thomas and J. Irvine. Connection and resource allocation of iot sensors to cellular technology-lte. *11th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, pages 365–368, 2015.

[70] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya. Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management. *8th international conference on network and service management (CNSM) and workshop on systems virtualiztion management (SVM)*, pages 385–392, 2012.

[71] D. Tychalas and H. Karatza. Simulation and performance evaluation of a fog system. *Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 26–33, 2018.

[72] Z. Wang, Z. Zhou, S. Jia, H. Liu, D. Li, and J. Cheng. Cloudeval: A simulation environment for evaluating the dynamic cloud vm consolidation. *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*, pages 37–45, 2016.

[73] B. Wickremasinghe, R. N. Calheiros, and R. Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 446–452, 2010.

[74] E. Winarno, W. Hadikurniawati, and R. N. Rosso. Location based service for presence system using haversine method. *International Conference on Innovative and Creative Information Technology (ICITech)*, pages 1–4, 2017.

[75] R. R. Yager and L. A. Zadeh. An introduction to fuzzy logic applications in intelligent systems. *Springer Science and Business Media*, 165, 2012.

[76] S. Yi, C. Li, and Q. Li. A survey of fog computing: Concepts, applications and issues. *In Proceedings of the Workshop on Mobile Big Data*, page 37–42, 2015.

[77] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[78] X. Zeng, S. K. Garg, P. Strazdins, P. P. Jayaraman, D. Georgakopoulos, and R. Ranjan. Iotsim: A simulator for analysing iot applications. *Journal of Systems Architecture*, 72:93–107, 2017.

[79] H. Zhang, Y. Xiao, S. Bu, D. Niyato, F. R. Yu, and Z. Han. Computing resource allocation in three-tier iot fog networks: A joint optimization approach combining stackelberg game and matching. *IEEE Internet of Things Journal*, 4:1204–1215, 2017.

Summary

This PhD thesis presents a conclusion of a six-year-long research in the field of Cloud Computing, Fog Computing, and Internet of Things. The main goal of the inter-cooperation of such domains, which are often associated with the Cloud-to-Thing Continuum is to process, store, and analyse vast amount of data of IoT applications in an effective way. The latest complex distributed systems involving thousands of IoT devices promote widely usable services by leveraging the computing and storing capacities of cloud data centres. To enhance the elasticity of a concrete service, cloud resources are often aided by resource-constrained fog nodes to improve the response time of the IoT application and to disperse the various types and unforeseen amounts of data.

These IoT-Fog-Cloud systems require significant investments in terms of design, development and operation, therefore, the use of simulators for their investigation is inevitable. There are a large number of simulators addressing the analysis of parts of these systems, however, it is obvious that only a state-of-the-art simulator is capable of modelling complex architectures in a realistic way, which meets modern challenges.

This PhD thesis consists of three theses separated into three major chapters. The first chapter presents a detailed survey and taxonomy of various IoT, cloud, and fog simulators in order to determine the key requirements of a compact and well-defined IoT-Fog-Cloud simulator. Furthermore, it presents an in-depth analysis and a comparison of two major fog simulators. The second chapter introduces the IoT and the pricing extension, exploiting a multi-cloud environment with resource allocation strategies in the DISSECT-CF-IoT simulator. Finally, in the third chapter, the DISSECT-CF-Fog simulator is presented which is able to model a multi-layered fog topology with energy measurement, task allocation algorithms and, mobility and actuator events.

Contributions of the Thesis

In **Chapter 2**, 44 simulation solutions modelling clouds, IoT and fogs were thoroughly analysed and classified. In our taxonomy, we separated the considered simu-

lators into three groups, and presented comparison tables and figures based on the taxonomy to reveal their differences and highlight how they model the elements of IoT-Fog-Cloud systems. We also compared two fog modelling approaches, namely iFogSim and DISSECT-CF-Fog. Finally, we presented an evaluation, which showed how to create and execute simulated IoT scenarios using fog and cloud resources with these tools.

The results of this chapter belong to **Thesis I**, and its contents were published in papers [P3], [P7], and [P11]. My contributions presented in this chapter are the following:

I/1. I proposed a comprehensive survey and taxonomy of numerous simulation approaches aiming at the examination of cloud, IoT, and fog systems.

I/2. I compared and analysed these simulation environments in terms of functionality, usability, maintainability, and code quality in order to determine the most relevant properties for modelling IoT-Fog-Cloud systems.

I/3. I presented an in-depth performance analysis with a comparison of the two most prominent simulators in these fields, namely DISSECT-CF-Fog and iFogSim, showing a significant performance difference in favour of DISSECT-CF-Fog.

In **Chapter 3**, we introduced the DISSECT-CF-IoT simulator, which is able to model generic IoT sensors, devices, and applications with detailed pricing schemes. Finally, we also proposed four resource allocation (i.e. cloud selection) strategies aimed to reduce IoT application execution time and usage costs for the multi-cloud environment.

The results of this chapter belong to **Thesis II**, and its contents were published in papers [P1], [P2], [P5], [P6], and [P10]. My contributions presented in this chapter are the following:

II/1. I investigated IoT-Cloud use cases to derive a general IoT use case based on meteorological forecasting, and I used it to evaluate the proposed IoT model.

II/2. I laid the foundations for the flexible and scalable modelling of IoT systems, and I implemented it in the DISSECT-CF-IoT simulator.

II/3. I analysed the operating costs of the meteorological IoT use case, and I developed a novel cost estimation extension using real cloud and IoT provider pricing schemes.

II/4. I introduced greedy and Pliant-based resource allocation strategies to reduce application execution time and utilisation costs for multi-cloud environments.

In **Chapter 4**, we introduced the DISSECT-CF-Fog simulator with a more detailed fog model to enhance its location awareness and multi-layer fog node management by introducing four task allocation strategies to manage task offloading decisions of complex IoT-Fog-Cloud systems. Finally, the simulation tool was extended towards modelling actuators and mobility features, and the energy measurements of IoT devices.

The results of this chapter belong to **Thesis III**, and its content was published in papers [P4], [P8], and [P9]. My contributions presented in this chapter are the following:

III/1. I designed a generic model of Fog Computing and implemented it in the DISSECT-CF-Fog simulator to enable the modelling of the Cloud-to-Thing Continuum.

III/2. I proposed greedy and Pliant-based task allocation algorithms for fog and cloud infrastructure management to optimise IoT application makespan, utilisation costs, and energy consumption.

III/3. I designed a realistic and dynamic IoT behaviour modelling supporting IoT mobility features, which can be configured by using the novel actuator interface of DISSECT-CF-Fog.

III/4. I designed an IoT energy model based on real-world experiments and proposed an extension of the DISSECT-CF-Fog simulator for the energy usage monitoring of IoT devices.

III/5. I also presented an enhancement of the basic weather forecasting IoT scenarios, towards modelling more refined IoT logistics and healthcare case studies.

Összefoglalás

Ez a doktori értekezés a Felhő és Köd Számítások (Cloud Computing and Fog Computing) és a Dolgok Internete (Internet of Things - IoT) területén végzett hatéves kutatást foglalja össze. Ezen területek - amelyekre gyakran Cloud-to-Thing Continuumként hivatkoznak - együttműködésének fő célja az IoT alkalmazások által kezelt hatalmas adatmennyiség hatékony feldolgozása, tárolása és elemzése. A legújabb, több ezer IoT eszközt magában foglaló összetett és elosztott rendszerek a felhő-alapú adatközpontok számítási és tárolási kapacitásának kihasználásával széles körben használható szolgáltatásokat biztosítanak. Egy konkrét szolgáltatás rugalmasságát növeli, ha a felhő rendszerek erőforrásait gyakran erőforrásaiban korlátozott köd csomópontok segítik, az IoT-alkalmazás válaszidejének javítása és a különböző típusú, akár előre nem látható adatok kezelése érdekében.

Ezek az IoT-Köd-Felhő rendszerek jelentős beruházásokat igényelhetnek a tervezés, a fejlesztés és az üzemeltetés tekintetében, ezért elkerülhetetlen a szimulátorok használata a vizsgálatukhoz. Számos szimulátor foglalkozik ezen rendszerek egyes részeinek elemzésével, azonban csak egy korszerű szimulátor képes a komplex architektúrák realisztikus, a modern kihívásoknak megfelelő modellezésére.

A doktori értekezés három tézisből áll, amelyek három fő fejezetre osztják a disszertációt. Az első fejezet a különböző IoT, felhő és köd szimulátorok részletes áttekintését és osztályozását mutatja be annak érdekében, hogy meghatározzuk egy kompakt és jól definiált IoT-köd-felhő szimulátorral szemben támasztott legfontosabb követelményeket. Továbbá bemutatja két jelentősebb köd szimulátor alapos elemzését és összehasonlítását. A második fejezet részletezi az IoT és az árazási kiterjesztést, kihasználva a több felhőből álló környezetet a DISSECT-CF-IoT szimulátorban. Ezen túl különböző erőforrás-elosztási stratégiákat is bemutatunk. Végül, a harmadik fejezetben a DISSECT-CF-Fog szimulátor kerül bemutatásra, amely képes egy többrétegű köd topológia modellezésére mobilitás és aktuátor események támogatásával. A szimulátort bővítettük különféle feladatkiosztási algoritmusokkal és az IoT eszközök energiafogyasztását mérő funkcióval is.

A tézisek kontribúciói

A **2. fejezetben** összesen 44 darab, felhő, IoT és köd rendszereket modellező szimulációs megoldást elemeztünk és osztályoztunk. A vizsgált szimulátorokat három csoportra osztottuk, és az osztályozás alapján összehasonlító táblázatokat és ábrákat mutattunk be, hogy feltárjuk a különbségeket, és rávilágítsunk arra, hogyan modellezik az IoT-Köd-Felhő rendszerek egyes elemeit a különböző megvalósítások. Kettő, köd rendszereket modellező megközelítést részletesebben összehasonlítottunk, nevezetesen az iFogSim és a DISSECT-CF-Fog szimulátorokat. Végül bemutattuk, hogyan lehet ezekkel a szimulációs eszközökkel köd és felhő erőforrások használó IoT használati eseteket létrehozni és elemezni.

Ebben a fejezetben bemutatott kutatás az **I. tézishez** tartozik, és a [P3], [P7] és [P11] publikációkban kerültek részletesebben tárgyalásra. A fejezetben bemutatott fő hozzájárulások a következők:

I/1. Kidolgoztam egy taxonómiát és rendszerezést a felhő, IoT és köd rendszerek vizsgálatát célzó számos szimulációs megközelítés összehasonlításához.

I/2. Elemeztem és összehasonlítottam a vizsgált szimulációs környezeteket funkcionalitás, használhatóság, karbantarthatóság és kódminőség szempontjából annak érdekében, hogy meghatározzam az IoT-Köd-Felhő rendszerek modellezéséhez leginkább releváns tulajdonságokat.

I/3. Bemutattam egy mélyreható teljesítményelemzést a két legjelentősebb szimulátor, nevezetesen a DISSECT-CF-Fog és az iFogSim összehasonlításával, amely jelentős teljesítménykülönbséget mutatott a DISSECT-CF-Fog javára.

A **3. fejezetben** bemutattuk a DISSECT-CF-IoT szimulátort, amely képes részletes árképzési sémákkal modellezni az IoT szenzorokat, eszközöket és alkalmazásokat. Végül négy erőforrás-elosztási (azaz felhő választási) stratégiát is javasoltunk a többfelhős környezethez, amelyek célja az IoT alkalmazások végrehajtási idejének és költségeinek csökkentése.

Ebben a fejezetben bemutatott kutatás a **II. tézishez** tartozik, és a [P1], [P2], [P5], [P6] és [P10] publikációkban kerültek részletesebben tárgyalásra. A fejezetben bemutatott fő hozzájárulások a következők:

II/1. Megvizsgáltam különböző IoT-felhő felhasználási eseteket, amelyek alapján meghatároztam egy meteorológiai előrejelző rendszereken alapú általános IoT felhasználási esetet, amit aztán a javasolt IoT-modell kiértékelésére használtam.

II/2. Kidolgoztam az IoT rendszerek rugalmas és skálázható modellezésének alapjait, majd megvalósítottam azt a DISSECT-CF-IoT szimulátorban.

II/3. Elemeztem a meteorológiai IoT használati eset működési költségeit, és egy újszerű költségbecslési kiterjesztést dolgoztam ki valós felhő és IoT szolgáltatók árképzési szabályainak felhasználásával.

II/4. Kifejlesztettem különböző mohó és Pliant-alapú erőforrás-elosztási stratégiákat az alkalmazások végrehajtási idejének és költségeinek csökkentésére a több felhőből álló környezetek számára.

A **4. fejezetben** bevezettük a DISSECT-CF-Fog szimulátort köd rendszerek modellezésére, amely figyelembe veszi a köd csomópontok fizikai pozícióját is. A többrétegű köd architektúrák kezelésének javítása érdekében különböző alkalmazás stratégiákat vezettünk be a komplex IoT-Köd-Felhő rendszerek tehermentesítő döntéseinek kezelésére. Végül a szimulációs eszközt kiterjesztettük az aktuátorok és a mobilitási jellemzők modellezésére, valamint az IoT eszközök energiafogyasztásának mérésére.

Ebben a fejezetben bemutatott kutatás a **III. tézishez** tartozik, és a [P4], [P8] és [P9] publikációkban kerültek részletesebben tárgyalásra. A fejezetben bemutatott fő hozzájárulások a következők:

III/1. Megterveztem a köd rendszerek általános modelljét, és implementáltam a DISSECT-CF-Fog szimulátorban, hogy lehetővé tegyem IoT-Köd-Felhő rendszerek modellezését.

III/2. Kifejlesztettem mohó és Pliant-alapú feladatkiosztási algoritmusokat a köd és felhő infrastruktúra kezeléshez, az IoT-alkalmazások végrehajtási idejének, használati költségeinek és energiafogyasztásának optimalizálása érdekében.

III/3. Megterveztem egy realisztikus és dinamikus viselkedés modellt, amely támogatja az IoT mobilitási jellemzőit, és amely a DISSECT-CF-Fog újszerű aktuátor interfészének használatával konfigurálható.

III/4. Kidolgoztam egy valós világbeli kísérleteken alapuló IoT energia modellt, és ennek alapján javaslatot tettem a DISSECT-CF-Fog szimulátor kiterjesztésére az IoT-eszközök energia felhasználásának mérésére.

III/5. Bemutattam az általános meteorológiai előrejelző IoT használati eset továbbfejlesztését egy logisztikai és egy egészségügyi használati eset modellezéséhez.

Publications

Journal publications

[P1] **A. Markus**, G. Kecskemeti and A. Kertesz. Cost-aware IoT extension of DISSECT-CF. *Future Internet*, Volume 9, 2017. DOI: 10.3390/fi9030047

[P2] **A. Markus** and J. D. Dombi. Multi-Cloud Management Strategies for Simulating IoT Applications. *Acta Cybernetica*, Volume 24, 2019. DOI: 10.14232/actacyb.24.1.2019.7

[P3] **A. Markus** and A. Kertesz. A Survey and Taxonomy of Simulation Environments Modelling Fog Computing. *Simulation Modelling Practice and Theory*, Volume 101, 2020. DOI: 10.1016/j.simpat.2019.102042

[P4] **A. Markus**, M. Biro, G. Kecskemeti and A. Kertesz. Actuator behaviour modelling in IoT-Fog-Cloud simulation. *PeerJ Computer Science*, 7:e651, 2021. DOI: 10.7717/peerj-cs.651

Full papers in conference proceedings

[P5] **A. Markus**, G. Kecskemeti and A. Kertesz. Flexible Representation of IoT Sensors for Cloud Simulators (PDP). In *Proceedings of 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 199-203, 2017. DOI: 10.1109/PDP.2017.87

[P6] **A. Markus** and A. Kertesz. Simulating IoT Cloud Systems: A Meteorological Case Study. In *Proceedings of Second International Conference on Fog and Mobile Edge Computing (FMEC)*, 171-176, 2017. DOI: 10.1109/FMEC.2017.7946426

[P7] **A. Markus**, P. Gacsi and A. Kertesz. Develop or Dissipate Fogs? Evaluating an IoT Application in Fog and Cloud Simulations. In *Proceedings of 10th Interna-*

tional Conference on Cloud Computing and Services Science (CLOSER), 193-203, 2020. DOI: 10.5220/0009590401930203

[P8] **A. Markus** and A. Kertesz. Modelling Energy Consumption of IoT Devices in DISSECT-CF-Fog. In *Proceedings of 11th International Conference on Cloud Computing and Services Science (CLOSER)*, 320-327, 2021. DOI: 10.5220/0010500003200327

[P9] **A. Markus**, J. D. Dombi and A. Kertesz. Location-aware Task Allocation Strategies for IoT-Fog-Cloud Environments. In *Proceedings of 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 185-192, 2021. DOI: 10.1109/PDP52278.2021.00037

Further related publications

[P10] **A. Markus**, A. Marques, G. Kecskemeti and A. Kertesz. Efficient Simulation of IoT Cloud Use Cases. In *Autonomous Control for a Reliable Internet of Services*, 313-336, 2018. DOI: 10.1007/978-3-319-90415-3_12

[P11] **A. Markus** and A. Kertesz. Investigating IoT Application Behaviour in Simulated Fog Environments. In *Cloud Computing and Services Science*, 258-276, 2021. DOI: 10.1007/978-3-030-72369-9_11

"Performance is bounded, but success is unbounded."

— Albert-László Barabási

Acknowledgements

First of all, I would like to thank my supervisor, Dr. Attila Kertész, for directing me to the path of a researcher in 2016. Over the past years I could participate in research projects, review papers for scientific journals, supervise bachelor and master students, and I could present my research in workshops, conference talks, and student competitions. Without his useful critics, encouragement, and guidance this thesis would have never come true.

I am grateful to my colleagues at the Department of Software Engineering, to my co-authors and my friends for helping in realising my dissertation. Special thanks to Dr. Tamás Pflanzner and Csaba Fülöp that I could always count on them.

Last but not least, I cannot thank my parents, my sister and my aunt enough for their constant care and support. Thank you, Tina, from the bottom of my heart, for accompanying me on this adventure.

I could not have done it without you.

András Márkus, 2022