ABSTRACT

SINGH, ANAND. SLA based Service Differentiation in Cloud Data Centers. (Under the direction of Ioannis Viniotis.)

In this thesis, we investigate a problem in the realm of hosting Internet of Things (IoT) broker services in a cloud computing datacenter. IoT sensors generate measurements that are transported to the datacenter for processing. Several tenants, each with his/her own measurement generation patterns are housed in such a center. The tenants and datacenter provider enter a Service Level Agreement (SLA) that formally states the service expectations (tenant) and obligations (provider). In the particular SLA we have considered, the only expectation from the tenant is to "have a specific amount of measurements processed within a given time period"; the provider's obligation is to provide the CPU resources to meet this expectation regardless of the details of the measurement generation pattern. The main merit of this SLA is its (technical and legal) simplicity.

The main technical problem we address is how can this SLA be guaranteed? This is a question that the datacenter designer must answer. The designer has several tools at his/her disposal in answering this question - we call such tools "knobs" that can be introduced in algorithmic solutions. The main knob we focus on, in this work, is CPU time provisioning.

We tackle this technical problem by breaking it down to three smaller problems. First, we study the SLA in its present form. We formulate two problems; in the first, we assume that a single server can provide the needed time to meet the tenant expectation. The control knob is fairly simple to design in this environment. In the second, we consider the situation where several servers must be provisioned, in order to meet the expectation. In the third formulation of the problem, we modify the SLA -we remove some of its simplicity- in order to simplify the designed controls.

The main technical challenge in all problems when deciding how to provision CPU time is the lack of knowledge about traffic patterns. An additional challenge is present in the case of multiple servers - the presence of two conflicting objectives that come from the tenant and the cloud provider. The tenant objective can be expressed in terms of, say, acceptable average delays, while the provider's objective can be expressed in terms of, say, acceptable operating cost. Even though these objectives are not explicitly stated in the SLAs, it is to the best interest of all parties involved if the designer was to come up with resource provisioning controls that will keep both parties happy.

The main contribution of this thesis is twofold: (a) the *introduction of the new SLA*, and, (b) the *design and evaluation of the CPU provisioning algorithms* to meet the SLAs.

© Copyright 2017 by Anand Singh

All Rights Reserved

SLA based Service Differentiation
in Cloud Data Centers

by
Anand Singh

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Engineering

Raleigh, North Carolina

2017

APPROVED BY:

Harilaos G. Perros	Mihail L. Sichitiu
Do Young Eun	Ioannis Viniotis Chair of Advisory Committee

DEDICATION

To my parents, my wife Avani, and all my family members.

BIOGRAPHY

Anand Singh was born in India in 1985. He received his B.Tech. degree in Electronics and Communication Engineering from NIT Jaipur, India in 2008, and an MS degree in Computer Networking from North Carolina State University, Raleigh, USA in 2013. He worked in Cisco Systems in voice technology group from 2008 to 2011.

At NC State, Anand was a teaching assistant for all twelve semesters and instructed labs on switched network design and management, routed network design and management, internetworking, and Internet of Things. At NC State, in every summer, he led groups of students in their summer projects.

He is happily married to Avani since December 12, 2016.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my guru, my Aristotle, my advisor Prof. Yannis Viniotis for guiding my Ph.D. study and related research. He helped me not only to grow as a researcher and instructor but also made me a better person. The three life lessons, the seven fundamental truths in networking, how to think like a designer, love for fundamentals, and many more such lessons (including three P's) from him are going to help me in all walks of life. I feel blessed to have a mentor like him.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Harry Perros, Dr. Mihail Sichitiu, and Dr. Do Young Eun for their insightful comments and encouragement. I still remember Dr. Perros giving me the personal books on service management to encourage me to explore service concepts more fundamentally. That gesture made me emotional and more determined to keep learning. Thanks to Dr. Sichitiu, I was able to reduce my rough edges in explaining concepts. His comment "I want you to be like a sharp knife, when you cut a cake it should be perfect" inspired me to improve my sharpness in technical discussions.

My sincere thanks also go to Dr. Andy Rindos (For all the support from IBM cloud for my Ph.D. and the professional guidance throughout), Carlos Pignataro, Joe Clark, Skip Booth, Frank Brockners (Cisco Distinguished Engineers, for helping me with practical knowledge from industry and supporting my lab setup), and Arun Arunachalam (for collaboration on Wireshark NCSU-Cisco project, it helped me a lot to understand protocol design).

I thank my students and my summer group members for the stimulating discussions. I am thankful to Dr. Viniotis for giving me an opportunity to teach, guide and work with many students in EB2-2027. The countless discussion with my summer group and lab students broadened my technical horizon and made my belief stronger in teaching and learning together. Thank you all for the fun we had in past 6 years and for our bond of friendship for the lifetime.

Last but not the least, I would like to thank my family for their constant support during my Ph.D.

TABLE OF CONTENTS

LIST OF TABLES . . . . . **viii**

LIST OF FIGURES . . . . . **ix**

Chapter 1 Service Differentiation in Network Systems **1**
1.1 Background 1
1.1.1 Networks 2
1.1.2 Enterprise Datacenters 2
1.1.3 Cloud Computing Environments 4
1.1.4 Internet of Things 4
1.2 Services 4
1.2.1 Introduction 4
1.2.2 Service Level Agreements 7
1.2.3 Service Models in Cloud Computing 9
1.3 Service Differentiation 10
1.3.1 Criterion for Service Differentiation 10
1.3.2 Approaches for Enforcing Service Differentiation 10
1.4 Related Work on Service Differentiation 11
1.4.1 Service Differentiation in the Internet 11
1.4.2 Service Differentiation in Network Connectivity Services 12
1.4.3 Service Differentiation in Hosting Services 12
1.4.4 Service Differentiation in Cloud Services 13
1.4.5 Summary: the Big Picture 13
1.5 Dynamic Closed Loop Approach for Service Differentiation 13
1.5.1 The Generic Objective 13
1.5.2 The Elements of the Control Framework 14
1.6 Problems Considered in this Thesis 15
1.7 Contributions of the Thesis 15
1.8 Thesis Organization 16

Chapter 2 SLA for IoT Cloud Computing **17**
2.1 Internet of Things 17
2.2 IoT Cloud Computing 19
2.2.1 IoT Cloud Services 20
2.2.2 Service Requirements for IoT Cloud Computing 21
2.2.3 IoT Cloud SLA 22
2.3 Motivation for SLA-based Resource Allocation in IoT Cloud Environment 24
2.3.1 Need for IoT SLA-based Resource Allocation 24
2.3.2 How is volume-based IoT Cloud SLA Different from a Networking SLA? 24
2.3.3 Why is IoT SLA-based Resource Allocation a Challenging Problem? 25
2.4 Related Work in SLA-based Resource Allocation 25
2.4.1 SLA-based Resource Allocation in Classical Networks 25

2.4.2 SLA-based Resource Allocation in Cloud Computing 26
2.4.3 SLA-based Resource Allocation in IoT Cloud Computing 27
2.4.4 Differences in the Various Allocations 28

Chapter 3 System Model 29
3.1 Elements of the Physical System and System Model 29
3.2 Cloud Deployment Models 31
3.2.1 Single-tier Service Cloud Deployment 31
3.2.2 Multitier Service Cloud Deployment 32
3.3 IoT Applications in the IoT Datacenter 33
3.4 Resources in the IoT Datacenter 34
3.5 Controls in the IoT Datacenter 34

Chapter 4 Resource Allocation for Current IoT SLA - Single Service Instance 35
4.1 Problem Formulation 35
4.1.1 System Architecture 35
4.1.2 Formal IoT SLA 36
4.1.3 Definition of Key Terms 37
4.1.4 Objectives 38
4.2 Our Approach 38
4.2.1 Control Parameters and Constraints for Achieving SLA 38
4.2.2 Design Considerations 39
4.2.3 Assumptions 40
4.2.4 Algorithm to Achieve the Current SLA 40
4.3 Experimental Evaluation 42
4.3.1 Experimental Setup 42
4.3.2 Traffic Patterns 42
4.3.3 Evaluation of Proposed Solution 43
4.3.4 Overhead and Tradeoff Analysis 45
4.4 Conclusion 46

Chapter 5 Resource Allocation for a Novel IoT SLA - Single Service Instance 47
5.1 Need for New IoT SLA 47
5.2 Problem Formulation 48
5.2.1 System Architecture 48
5.2.2 A New IoT SLA 48
5.2.3 Definition of Key Terms 49
5.2.4 Objectives 50
5.3 Our Approach 50
5.3.1 Control Parameters and Constraints for Achieving SLA 50
5.3.2 Design Considerations 50
5.3.3 Assumptions 51
5.3.4 Algorithm to Achieve the New SLA 51
5.4 Experimental Evaluation 54
5.4.1 Experimental Setup 54

5.4.2 Traffic Patterns . . . 54
5.4.3 Evaluation of Proposed Solution . . . 54
5.4.4 Overhead and Tradeoff Analysis . . . 57
5.5 Conclusion . . . 57

Chapter 6 Resource Allocation for Current IoT SLA - Cluster of Service Instances . . . 59
6.1 Service Clusters in Cloud Computing . . . 60
6.1.1 Cluster of Service Instances in Cloud . . . 60
6.1.2 Server Scaling in Cloud Computing . . . 60
6.2 Related Work in Resource Scaling in Cloud Computing . . . 61
6.2.1 Reactive Approaches . . . 61
6.2.2 Proactive Approaches . . . 62
6.3 Problem Formulation . . . 63
6.3.1 System Architecture . . . 63
6.3.2 IoT SLA . . . 63
6.3.3 Definition of Key Terms . . . 63
6.3.4 Objectives . . . 66
6.4 Our Approach: Design of Scaling Policies to Achieve the SLA . . . 67
6.4.1 Introduction and Motivation . . . 67
6.4.2 Assumptions . . . 67
6.4.3 Specific Designs: Static Policies . . . 67
6.4.4 Specific Designs: Dynamic Policies . . . 70
6.4.5 Summary: the Selected Sets of Policies . . . 76
6.5 Experimental Evaluation . . . 76
6.5.1 Experimental Setup . . . 77
6.5.2 Traffic Patterns . . . 77
6.5.3 Evaluation of Policies Within a Given Class Only . . . 78
6.5.4 Answering Specific Design Questions . . . 89
6.5.5 Overhead and Tradeoff Analysis . . . 102
6.6 Conclusion . . . 102

Chapter 7 Conclusion and Future Work . . . 105
7.1 Thesis Conclusion . . . 105
7.2 Future Work . . . 106

BIBLIOGRAPHY . . . 110

LIST OF TABLES

Table 6.1 Configuration complexity of various policies. 102
Table 6.2 Measurement overhead of various policies. 103
Table 6.3 Bandwidth overhead of various policies. 103
Table 6.4 CPU processing overhead of various policies. 104
Table 6.5 Memory overhead of various policies. 104

LIST OF FIGURES

Figure 1.1 An architectural model for a datacenter with access gateways, layered switch fabric, and servers. 3
Figure 1.2 The feedback loop approach. 14

Figure 2.1 The IoT hype curve, according to Gardner. 18
Figure 2.2 A history of IoT. 19
Figure 2.3 An example of IoT cloud PaaS solution 20

Figure 3.1 Single tier service cloud deployment model. 31
Figure 3.2 Multitier service cloud deployment model. Preprocessing and service tier are two tier of service instances. 32
Figure 3.3 Traffic flows in a IoT cloud datacenter. 33

Figure 4.1 Topology considered: Single service instance (message server) for all tenants. 36
Figure 4.2 Total enforcement period. 39
Figure 4.3 SLA Conformance. 43
Figure 4.4 The effect of service capacity overprovisioning. 44
Figure 4.5 SLA Conformance in case of excess traffic. 45

Figure 5.1 Splitting the total enforcement period into K subperiods. 49
Figure 5.2 Measurement and enforcement time in a subperiod. 51
Figure 5.3 SLA Conformance. 55
Figure 5.4 SLA Conformance. 55
Figure 5.5 Effect of number of subperiods on additional enforcement time. 56
Figure 5.6 Effect of overprovisioning factor n on additional enforcement time. 57

Figure 6.1 Topology considered: Cluster of service instances (message servers) for all tenants. 64
Figure 6.2 Objective (O1) for different traffic patterns with policy p_1, favoring tenant. . . 80
Figure 6.3 Objective (O1) for different traffic patterns with policy p_2, favoring provider. . 81
Figure 6.4 Objective (O2) for different traffic patterns with policies p_1 and p_2. 82
Figure 6.5 Objective (O3) for different traffic patterns with policies p_1 and p_2. 83
Figure 6.6 Objective (O2) for different traffic patterns for static periodic policy p_4 with periods $K = 10, 20, 100$. 84
Figure 6.7 Objective (O3) for different traffic patterns for static periodic policy p_4 with periods $K = 10, 20, 100$. 85
Figure 6.8 Objective (O2) for different traffic patterns with dynamic policies p_6, p_7, and p_8 in $\mathscr{P}_{rule}$ with delay threshold $d_{th} = 60$. 87
Figure 6.9 Objective (O3) for different traffic patterns with dynamic policies p_6, p_7, and p_8 in $\mathscr{P}_{rule}$ with delay threshold $d_{th} = 60$. 88
Figure 6.10 Effect of N on Objective (O3) for different traffic patterns, policy p_7. 90
Figure 6.11 Effect of N on Objective (O2) for different traffic patterns, policy p_7. 91

Figure 6.12 Effect of N on Objective (O3) for different traffic patterns except single burst, policy p_7. 92
Figure 6.13 Effect of d_{th} on Objective (O2) for traffic pattern 'burst at every $T/24$'. 94
Figure 6.14 Effect of d_{th} on Objective (O2) for traffic pattern 'burst at every $T/100$'. 95
Figure 6.15 Effect of d_{th} on Objective (O3) for traffic pattern 'burst at $T/24$'. 96
Figure 6.16 Effect of d_{th} on Objective (O3) for traffic patterns 'burst at $T/100$'. 97
Figure 6.17 Effect of d_{th} on Objective (O2) for different traffic patterns, policy p_7. 98
Figure 6.18 Effect of d_{th} on Objective (O3) for different traffic patterns, policy p_7. 99
Figure 6.19 Effect of d_{th} on Objective (O2) for different traffic patterns, policy p_7. 100
Figure 6.20 Effect of d_{th} on Objective (O3) for different traffic patterns, policy p_7. 101

CHAPTER

1

SERVICE DIFFERENTIATION IN NETWORK SYSTEMS

The chapter is organized as follows. In Section 1.1, we describe enterprise datacenters, organized as a cloud computing facility and hosting Internet of Things applications. In Section 1.2, we introduce the concept of Service Level Agreements. In the next section, we present how such agreements are used to provide different types of service to the applications. In Section 1.4, we briefly outline how this service differentiation has been accomplished in various networking systems, from the early Internet to ISP to datacenters to present day cloud hosting environments. In Section 1.5, we discuss open and closed loop controls, the approaches at the heart of differentiated service designs. In Section 1.6, we briefly outline the three main problems we consider in this work. In the last two sections, we summarize our contributions and present the organization of the thesis.

1.1 Background

In this section, we give a brief overview of an enterprise datacenter, cloud computing, and Internet of Things (IoT). The main point of this section is to set a high-level background for the remainder of the thesis.

1.1.1 Networks

A network is connected system of 'nodes'. The nodes here can be processing (servers), storage or forwarding elements (routers or switches). Networks are usually categorized based on the component type, geographical area, communication model, applications and services. For example, a Local Area Network (LAN) and a Wide Area Network (WAN) can be differentiated in terms of geographical size. A LAN is "A data network intended to serve a small area"; a WAN is a data network covering a large geographic area. Of interest to us in this thesis are the LANs used inside datacenters. The two most popular LAN technologies today are Ethernet (based on IEEE802.3 [55]) and Wireless LAN (based on IEEE 802.11 [56]); they differ primarily in terms of their protocol stack and communication models. We mention networks here because of the service differentiation models developed for them, as we discuss further in Section 1.3.

1.1.2 Enterprise Datacenters

An enterprise datacenter (EDC) is a physical complex housing primarily physical servers, along with network infrastructure (network switches and routers, network service appliances) and (most of the time) separate storage devices [61]. The purpose of EDC is to provide enterprise services like application, compute, storage, etc.

A typical EDC architecture model is shown in Figure 1.1. The model has the following block components: (a) Gateway; this contains routers that connect the datacenter to the outside world (e.g., an ISP network), (b) Switch fabric (SF); this networks uses switches/routers to interconnect gateways, servers and storage, (c) Server tier; this is where applications and services run. In some architectures, the server tier can be divided into a preprocessing and main processing tier, (d) Storage tier; in some datacenters, this is a separate tier with storage devices and storage management applications., providing storage services to the server tier.

Scalability is a key design requirement in datacenters. In the server and storage tiers, scalability is simply achieved by adding more physical servers and storage devices. Scaling the SF network is achieved by modular and hierarchical topologies that are more sophisticated [62]. There are two main, popular SF topologies in use today. The leaf-spine has a leaf tier to connect severs and a spine tier to connect to gateways. The fat tree topology has an access tier to connect servers and an aggregation + a core tier to connect to gateways. Employing a larger number of layers decreases delay performance (hence, no designs are used with four or more layers) but offers greater scalability in terms of server connectivity.

The server tier is the "heart" of a datacenter offering compute, storage, and application services. Its architecture depends upon the nature of servers, the types of application, services and traffic

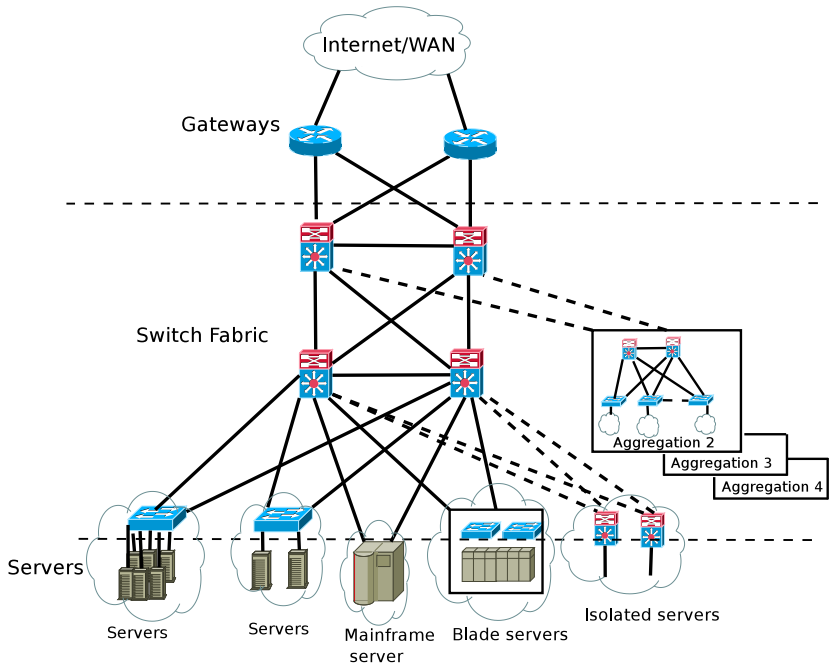

Figure 1.1 An architectural model for a datacenter with access gateways, layered switch fabric, and servers.

patterns. Virtual instead of physical servers offer efficiency and flexibility at the cost of performance. Virtualization offers the datacenter admin additional control possibilities (and headaches) with tools such as VM placement, VM scaling, and VM migration.

Applications deployed on (virtual or physical) servers have, in general, different requirements. For that reason, they are deployed in standalone, distributed, and parallel fashion. The mode of deployment impacts the datacenter design both at server as well as the SF tier. For example, an application distributed across two or more physical servers requires bandwidth across the SF (aka East-West bandwidth) to pass data between the servers; there is no such requirement if the application runs on two VMs collocated in the same physical server or if the application is standalone. An example of a distributed can be a search application where a web request comes to a proxy server then forwarded to one or more servers to the actual data source (e.g., a database server).

Management of a large datacenter is a challenging task; it is more complicated than, say, an ISP network or IT administrator's job. The datacenter administrator has to manage server, storage as well as networking components inside the datacenter. The management plane consists of network/server/storage management software and network/server appliances; for example a VSphere management suite [86] to manage ESXI-based virtualized servers.

A large component of the operation costs of a datacenter comes from the power and cooling

infrastructure; datacenter operators look for all possible ways to be energy-efficient. On the business side, these ways include decisions to house datacenter in cooler places (like Scandinavian countries or caves) to save on the cooling bill; or states/counties with subsidized electricity bills. On the research side, we are witnessing an avalanche of energy-efficient resource provision techniques to improve operation costs [21], [22], [80].

1.1.3 Cloud Computing Environments

In this section, we provide a generic introduction of cloud computing. The main point is to differentiate the cloud datacenter from EDC and provide a brief list of important challenges in the cloud datacenter.

We have seen many definitions of cloud computing in the literature. The authors in [85], [44] list more than twenty different ways researchers have defined cloud computing. We present here one widely accepted cloud definition given by NIST [66]: *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

For the purposes of this thesis, the key difference between cloud and enterprise datacenter is multi-tenancy. The latter houses one tenant only; the former is shared among a multitude of tenants. Note that the tenant is an administrative entity based on whose behalf the resources and services are managed by the provider. This creates additional challenges in the cloud case. The fundamental (and by no means simple) one is tenant identification.

1.1.4 Internet of Things

We give an introduction to the Internet of Things in Section 2.1. In the remainder of the thesis, an enterprise running an IoT system will share (with other tenants) a cloud datacenter for its processing needs.

1.2 Services

1.2.1 Introduction

In this section, we define the term services, describe its characteristics and present a framework for service management.

1.2.1.1 Service Definition

There are many definitions of services [95], [49], [72], [41]. We follow the generic definition of service from ITIL [9]: "Service is a means of delivering value to customers by facilitating outcomes customers want to achieve without the ownership of specific costs and risks". In simpler terms, service can be defined as "Work done by one person or group for the benefit of another person or group" [70]. In our daily life, we experience many services around us; for example, postal services, transport services, medical services, etc. which are offered by government and businesses to the users.

In the IT sector, we have seen many services in the past and new services are being introduced at rapid pace. For example, POTS (Plain Old Telephony Service) a voice-grade telephone service was popular in the past. It has been replaced by new voice services with additional offerings to a customer like voice, data, and video. Few other examples of IT services are web-hosting (hosting customers website), managed storage services (for example Dropbox, Google Drive for customers), content distribution (for example CDN providers distributing content of another business at multiple points of presence (POP) sites)), and Virtual Private Network (ISP offering business a private network service without geographical limitation).

In all the definitions and example services given above, there is a notion of co-production with the customer (that is, customers provide requirements, data, and their information to get access to the service) as well as "intangibility" (that is, customers do not know the physical presence of the service). For example, in a VPN service, the customer doesn't know the presence of the VPN routers or routing mechanisms but accesses this service by providing its information and data to ISPs. In the case of a non-service offering, the production takes the form of tangible "goods". Such goods are produced without customer input. We can take a web server and web hosting as an example to differentiate between non-service and service offerings. A Web server is a product already designed by a provider without input from the customer. Whereas in web hosting, the provider has the infrastructure and customer provided its identification and content to get a hosted website service.

1.2.1.2 Service Characteristics

A service has the following major characteristics; we explain them with a cloud computing example:

- Provider - the entity which provides service; for example, a cloud service provider like Google, Amazon, etc.
- Customer - the entity which consumes the service. for example, individuals and businesses consume compute infrastructure from cloud providers.

- Co-production - the provider creates and maintains the capability to offer a service. The customer provides information like customer ID, compute instance details (number, type of servers needed, duration etc.) in a co-production phase to create the compute service.

- Perishable nature of service - services can not be stored, reused or resold. For example, a compute instance created for a customer for a service duration cannot be resold to another customer.

- Degree of customization: customization of a service requires close interaction between provider and customer. A highly-customized service can incur huge costs. For example, a customer need closer interaction with a cloud provider if it requires a VM size not offered in the service catalog.

1.2.1.3 Service Management

ITIL is a popular framework for service management [40]. According to their philosophy, (service) management must occur "from cradle to grave" - throughout the entire lifecycle. Therefore, IT services must be planned, financed, delivered, operated and phased out. Their publication includes guidelines for five distinct phases, namely, Service Strategy, Service Design, Service Transition, Service Operations and Continual Service Improvement. Various vendors have started implementing their own ITIL-based framework with their own methods and software for IT service management. Two examples are IBM's Process Reference Model for IT (PRM-IT) [39] and Microsoft's Operating Framework (MOF) [71].

Following we give an example of ITIL stages in cloud services life cycle.

- Service strategy - a cloud provider needs to understand customer needs and decide on a strategy. In the strategy phase, the provider needs to address the following major points: what services to offer (for example, one of compute, database, analytics, visualization? all of them?), who are potential customers (individuals, large businesses requiring storage space, an IoT customer requiring heterogeneous databases?), who are the competitors, what is the revenue strategy?

- Service design - in this phase, the cloud provider starts with high-level business requirements (BRs) from the service strategy phase and ends up with technical requirements that meet the BRs. The provider produces technical design documents for the service features (X-as-a-service, where X can be compute, storage, application), service level objectives (means of measuring services and metric Key Performance Indicators), service level agreements

(discussed later in the chapter) needed with the customer, along with design considerations for service reliability, isolation, scalability etc.

- Service Transition - this is an implementation phase in which design documents are converted into implementation plans. Once all aspects of services are implemented as described in the design documents produced in service design phase, the services are made available for testing and operational use.

- Service Operation - in this phase, the cloud provider focuses on operations; that is, execution and consistent delivery of services.

- Continual Service improvement - this phase is concurrent with the operations phase. The idea is to monitor the service level and use this feedback to exercise controls to improve services. As an example, a cloud provider may be measuring bottleneck link or server utilization for the purpose of adjusting capacity plans.

1.2.2 Service Level Agreements

According to ITIL, "a Service Level Agreement (SLA) is an agreement between service provider and customer. The SLA describes the service, service level targets, measurement methods, Key Performance Indicators (KPI) and specifies the responsibilities of the service provider and customer". The SLAs are designed in the Service Design phase. If in the continual service improvement phase we change or improve the service, the original SLA should be changed accordingly.

Following is an example of a commercially-offered SLA by Amazon for Elastic Compute Cloud (EC2) services [4]. For simplicity, we present the SLA in terms of the ITIL definitions above.

- *Service*: the Amazon EC2 service is defined in [6]

- *Service Level Target*: targets are associated with Service Level Objectives (SLO). For this service, the SLO is defined as "Monthly Uptime Percentage" (MUP). MUP is the KPI mentioned in the SLA and calculated by subtracting from 100% the percentage of minutes during the month in which Amazon EC2 was in the state of "Region Unavailable". ("Unavailable" is defined as all of customer running services instances having no external connectivity.) A target may be, for example, a MUP value of at least 99.95%. This is the "Service Commitment" on behalf of the provider.

- *Responsibilities of the provider*: in the event Amazon does not meet the Service Commitment, the customer will be eligible to receive a "Service Credit", credited back to an eligible account

(not cash). The credit is a dollar credit, calculated as follows. If MUP is less than 99.99% but equal to or greater than 99.0%, service credit is 10% of the monthly bill; if MUP is less than 99.0%, then the service credit is 25%.

- *Responsibilities of customer*: a credit request must include the dates and times of each unavailability incident of the claim, the affected EC2 instance IDs and request logs that document the errors and corroborate the claimed outage.

1.2.2.1 Why are SLAs useful?

SLAs are useful to both provider and customer. For a customer, a SLA sets very clearly the expectations in terms of service offering and service level objectives. It also states the customer responsibility to measure the conformance of SLA with given KPIs in SLO. For a provider, a SLA sets very clearly the service offering targets which provider needs to conform to. It also states the responsibility of provider in case of non conformance.

SLAs are at the core of the problems we study in this thesis.

1.2.2.2 Why is defining service level targets a difficult task for the provider?

Defining service level targets is a challenging task as the provider needs to define the following items as precisely as possible:
a) *Customer expectations from the service*. Sometimes the desired or meaningful service level targets are not known in advance.
b) *Responsibilities of customer and provider*. For a customer, the responsibility includes measurement and reporting non-conformance. For a provider, the responsibility includes measurement and enforcement of service targets and defining the non-conformance policy. For both entities, these may be tasks that are too expensive or even impossible for KPIs that are not selected in a quantifiable manner.

As discussed earlier, ITIL, MOF, and PRM-IT all define a framework for carefully defining service levels. However, none of the frameworks is a widely accepted standard.

1.2.2.3 SLA: Measurement and Enforcement Concepts

Two basic concepts are associated with a SLA. Observing that the service level target is met (Measurement) and taking actions to guarantee that the targets are met (Enforcement). Note that measurements are taken by both entities, since they both need to decide whether the SLA is met. The

provider may take additional measurements, for the sake of deciding how to guarantee the SLA. Enforcement is solely a provider task.

A *measurable* SLA is loosely defined as one in which the KPIs can be easily and/or inexpensively observable by both provider and customer. In this thesis, we consider a SLA with a measurable KPI, namely "number of messages processed within a month". A non-measurable SLA is, therefore, one in which the measurements for KPIs may be hard to obtain or interpret. Consider, for example, the percent of uptime (as defined earlier in the Amazon EC2 SLA). A customer needs to design a complex measurement mechanism that keeps a log of external connectivity on all its instances "all the time". This can be quite expensive, to the point customers may choose not to monitor the SLA!

The term "Enforcement of SLA" describes all actions a provider can take so that service objectives are met. For example, an action may be adding another server if a performance-defined KPI is below the target mentioned in the SLA. Or, removing a server if the target is met. In the more technical jargon of subsequent chapters, enforcement encompasses the control algorithms designed to meet the objectives and the "knobs" available to fine tune the operation of an algorithm. Enforcement can be difficult or expensive for a provider, due to several reasons. We discuss such reasons later, when we build some additional background.

1.2.3 Service Models in Cloud Computing

Over the years, several different models of cloud computing have emerged in response to specific needs of the user. The following three are the main service models in cloud computing.

1. Infrastructure as a Service (IaaS): this offering includes the capability to provision "infrastructure" (that is, processing, storage, and network) resources.

2. Platform as a Service (PaaS): this offering includes the capability to deploy and manage customer-created applications.

3. Software as a Service (SaaS): this offering includes use of provider-supplied applications.

SLA related challenges in current Cloud: Apart from generic challenges of SLA definition, monitoring and enforcement discussed earlier, the cloud provider faces following additional challenges:

In a cloud datacenter that offers multiple service models, multiple SLAs are required. For example, in an IaaS service model, KPIs are different for a Storage SLA (generally expressed in storage-related units like number of data records, read/writes per second, etc.) than for a processing SLA (generally expressed in CPU-related units like CPU cycles, processing capacity for volume of messages, response time for processing a message, etc.). In a SaaS model, a CPU-related metric may be used. The cloud provider has a difficult task in designing and managing multiple SLAs.

Multitenancy makes managing SLAs significantly more difficult for the cloud provider. As different customers have different needs from the cloud service, the provider may need further per customer classification of SLAs within each service model. Multi tenancy also requires per tenant measurement and enforcement, adding complexity to the task.

1.3 Service Differentiation

As we just mentioned in the previous section, different customers have different needs from the cloud service. In industry speak, meeting such needs is Service Differentiation.

The classical Internet was designed with the so called "best effort connectivity service" as an objective. The classical Internet did not provide service differentiation, as all traffic was handled in the same fashion.

Service differentiation implies specification of different service level targets and SLOs (different service levels for short). For example, in a network connectivity service, if two packets are being buffered and scheduled differently inside a router, we create two differentiated service levels.

The requirement for service differentiation can come for both customer and provider as discussed in the next section.

1.3.1 Criterion for Service Differentiation

There are two sources of criteria for service differentiation:

- *Customer supplied Criteria.* Examples include: a) Content type, such as text or video, b) Data quality, for example, short lived vs long lived data, and, c) Service quality, for example, best effort vs real-time delivery service.

- *Provider supplied Criteria.* Examples include: a) Profit, for example, paying vs non paying customers (using promotional reason), and, b) Resource usage, for example, the provider decides to distribute additional resources for customers with best-effort requirements.

1.3.2 Approaches for Enforcing Service Differentiation

Two different service levels may involve different SLOs or different target values. Enforcement of a SLA in either case must guarantee *specific KPIs*. In general, there are two main approaches to control any KPI:

1. *Open Loop* - this approach is static in nature and doesn't require measurements from the system. For example, adding a new server in predefined times, in order to improve the response time KPI.

2. *Closed Loop* - this approach is dynamic in nature and requires measurements from the system. For example, measuring server queue size, and adding another server if it increases above a threshold, in order to improve the response time KPI.

Overall, an Open Loop approach is a simple mechanism to implement but offers less granular control to enforce an SLO. Whereas a Closed Loop approach is more complex to implement with additional measurement overhead but offers more granular control to enforce an SLO.

1.4 Related Work on Service Differentiation

In this section, we survey the related work in four areas within the network system and services space. The main point of this section is to describe service characteristics and service differentiation levels in classical Internet, services in network connectivity, web hosting and cloud computing. Both service characteristics and service level targets are given in the respective SLAs.

1.4.1 Service Differentiation in the Internet

The original Service model in the Internet was the "Best effort service model" [35]. In this model, there was no service differentiation at all, as all traffic was handled in same fashion.

As adoption of Internet started at rapid pace in early 90's, applications beyond email and file transfers were sending traffic - voice, video, teleconferencing, web, etc. Clearly, the needs of the users became widely different. There was a natural pressure to extend the best effort model [3]. As a result, an extended service model of the Internet came into the picture. To support it, two new architectures were proposed.

IntServ Architecture. [29] The Integrated Service (IntServ) architecture used a closed loop mechanism to control the SLO of the extended service model. The mechanism requires call admission, resource reservation, call admission, packet classification, scheduling and dropping function as "knobs" [91], [76], [30]. Due to the complexity of dynamic closed loop in resource reservation, this architecture was not widely accepted and dropped in favor of the Diffserv architecture.

DiffServ Architecture. [23] The Differentiated Service (DiffServ) architecture used an open loop mechanism to control the SLO of the extended service model. DiffServ is implemented in a per-hop basis which makes it simple to implement even if some routers in the network ignore or are not DiffServ capable.

Two major knobs in Diffserv are the traffic classifier and traffic conditioner. The traffic classifier classifies a packet in a traffic stream based on the content of some fields in the packet header. These classifiers are used to steer a packet to a suitable traffic conditioner. The traffic conditioner acts on the traffic profile (SLO defined in SLA) and contains the following knobs: Meter (used to measure a traffic stream against a traffic profile), Marker (to mark a packet which can be useful to select operation on the packet), Shaper (shapes the traffic to conform to a traffic profile), Dropper (drop a packet to conform to a traffic profile).

With the above controls in place, DiffServ provided a mechanism through which a packet that arrives at a router inside a DiffServ capable router can be steered into different buffers where different scheduling policies can be applied to achieve the desired SLO.

1.4.2 Service Differentiation in Network Connectivity Services

Connectivity is a service offered by an Internet Service Provider (ISP) to customer to connect over Internet. Customer requires connectivity services from an ISP for two reasons: a) to connect all their private sites with a WAN, and, b) to get on the public Internet.

ISPs have two types of service model to meet the customer requirement.

- VPN service: connects customer sites via an L2/L3 Virtual Private Network (VPN). Typical KPIs in a typical VPN-based SLA include number of sites connected, bandwidth, mode of connectivity, and reliability. An example VPN SLA from Sprint [67] uses an MPLS backbone network and includes the following KPIs: average end-to-end delay (in milliseconds), average end-to-end packet loss (0.1% to 0.55% depending on the locations of the eligible customer sites), site availability (up to 100% site availability) and refund policy in terms of service credit for non-conformance. The averages are computed over a monthly period.

- Internet Services: connects the customer to the public Internet. KPIs in a typical Internet Service SLA include availability, network latency, and packet loss. An example AT&T SLA [13] provides broadband service with the following KPIs: network availability (99.9%), network latency (40 milliseconds), packet loss (0.1%), and restoration period (1 day).

1.4.3 Service Differentiation in Hosting Services

We mention service differentiation in hosting (web) services for historical reasons only; nowadays, cloud offerings include it. In the past, such services were differentiated in two main forms: managed or unmanaged [87]. The criterion for differentiation mentioned in the SLA is typically the uptime and performance of the servers that are being hosted. As providers have no control over client traffic

or network (Internet in this case), the SLAs usually specify the performance in terms of specific server features. For example, the SLA states that a server is able to handle 1,000 request coming over one hour.

Common examples of Web hosting companies are GoDaddy [90] and Dreamhost [89].

1.4.4 Service Differentiation in Cloud Services

The various service models for datacenter and clouds have been mentioned in Section 1.2.3. A specific example of an SLA for the IaaS model is Amazon EC2 SLA discussed earlier. The service differentiation criterion mentioned in the SLA is percent uptime over a month.

Another specific example of an SLA, this time for the PaaS model, is the Google App Engine Service Level Agreement (SLA) [11]. The service differentiation criterion mentioned in this SLA is again percent uptime over a month.

1.4.5 Summary: the Big Picture

Service Differentiation has been employed in several contexts both in networking as well as datacenter environments. It has been implemented via open (static) as well as closed-loop (dynamic) approaches. We also apply both in the context of the IoT environment that is of interest in this work. The closed-loop approach has some additional advantages, as we explain in Chapter 6, so we provide some additional background on it next.

1.5 Dynamic Closed Loop Approach for Service Differentiation

In this section, we describe the elements of the feedback-based control model at a high-level. The scaling policies we design in Section 6.4.4, page 70, follow this model.

1.5.1 The Generic Objective

The generic objective of the feedback-based control approach, in the context of service differentiation is to meet any given KPI.

For example, in the Sprint SLA, one KPI is for monthly packet loss to be kept below a 0.1%. A feedback-based control based on forwarding path selection could work as follows: the daily loss is measured (feedback) and the loss from the beginning of the month up to date is calculated. If it exceeds the desired threshold, a “better” path is selected. Another example is the volume-related KPI in the SLA we study in Chapter 6.

We have discussed the merits of the closed loop approach in Section 1.3.2.

1.5.2 The Elements of the Control Framework

The elements of the closed loop framework are best described with the aid of Figure 1.2.

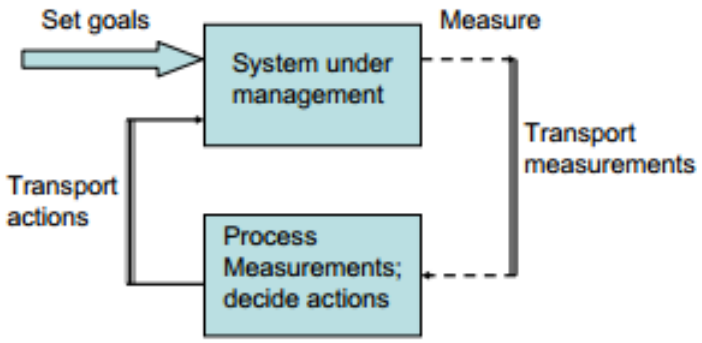

Figure 1.2 The feedback loop approach.

Set Goals. In our problem, the goal is to meet the (volume) KPI in the SLA.

System Under Management, (SUM). This is the system for which the KPI is set; in our problem, this is the server or servers which will process the IoT sensor data. In general, in an IoT environment, the "system" contains the actual, physical things and the sensors and/or actuators associated with it. *SUM and the IoT system should not be confused.*

Measurements. In general, the term refers to monitoring the goal; they are indications of whether the goal is met. Note that in the IoT system, *the term measurements is overloaded*; it also refers to the outputs of the sensors. In our problem, these are the messages that the servers must process. The precise meaning is clear from the context.

Process Measurements. Processing of the sensor measurements occurs in servers housed in a (cloud) datacenter environment. As a result of this processing, actions are decided and command signals are sent to the actuators housed in the SUM. Processing of the control-related measurements is done inside a dedicated controller that is separate from the message servers. In our problem, the actions change how CPU time is allocated at the server(s).

Transport Measurements. When the sensors and the servers are in geographically different

locations, measurements must be transported to the servers via a network. This transfer can be done under any application layer protocol; MQTT [68] and CoAP [27] are two standardized protocols designed particularly for IoT environments. When the controller and the points at which the goal is monitored are not in the same server, the goal-related measurements must also be transported to the controller. There is not standardized protocol for this transfer.

Transport Actions. A similar discussion applies to transferring the two types of actions to the IoT actuators (via MQTT or CoAP, for example) or the servers under control. △

Two major design issues in the feedback-based control approach are: (a) selection of what element of the SUM should be controlled (the control point), and, (b) selection of the type and frequency of measurements. We discuss them extensively in Section 6.4.4, page 70.

1.6 Problems Considered in this Thesis

The generic problem we consider in this thesis deals with SLA-based service differentiation in an enterprise datacenter. The specific SLA we consider has a single KPI, namely volume of (IoT) messages processed.

We break this problem into three smaller problems. First, we study the SLA in the form that is presently offered by several major cloud providers. We formulate two problems; in the first, we assume that a single server can provide the needed time to meet the tenant expectation. The control knob is fairly simple to design in this environment, using a static approach. In the second, we consider the situation where several servers must be provisioned, in order to meet the expectation. In this environment, dynamic approaches fare better than static ones, as we will see. In the third formulation of the problem, we modify the SLA KPI in order to simplify the designed controls.

1.7 Contributions of the Thesis

We claim two main contributions in this thesis: (a) the *introduction of the new SLA*, and, (b) the *design and evaluation of the CPU provisioning algorithms* to meet the SLAs. We introduce and justify the need for the new SLA in Section 5.2.2, page 48. The algorithms we design are presented in Sections 4.2.4, page 40, for the single service instance problem/current SLA, 5.3.4, page 51, for the single service instance problem/current SLA, and Section 6.4, page 67, for the multiple service instance problem/current SLA breakdowns.

1.8 Thesis Organization

In this chapter, we have presented the necessary background for the remaining parts of the thesis. In Chapter 2 , we present SLAs for IoT computing in cloud environments; we survey related work in the SLA-based resource allocation in Section 2.4. In Chapter 3, we present the system model and discuss its main elements, deployment scenario, resources, and constraints. The model provides the necessary framework to formulate the generic resource allocation problem we study in this thesis. We break the problem into three parts. In Chapter 4, we consider SLAs in their current form. Our assumption in this chapter is that a single instance is used to guarantee the SLA. In Chapter 5, we propose a new SLA and provide a server (CPU time) provisioning algorithm to meet it. In Chapter 6, we again consider SLAs in their current form. The difference with Chapter 4 is that here we consider multiple instances of server to guarantee the SLA. In Chapter 7 we conclude the thesis and suggest directions for future extensions.

CHAPTER

2

SLA FOR IOT CLOUD COMPUTING

The chapter is organized as follows. In Section 2.1, we briefly introduce IoT. In the next section, we present cloud computing models for IoT applications. The main point is the introduction of the IoT SLA in Section 2.2.3. This is the main subject of this work; we give examples of how several cloud providers state this SLA. In the next section, we provide the motivation these providers have to provision their datacenter resources in order to meet this SLA and conclude with a brief description of the main challenges they face in enforcing the SLA. Finally, in Section 2.4, we review relevant literature.

2.1 Internet of Things

The hype and predictions about the "Internet of Things" have started becoming real now (see Figure 2.1). We have started experiencing the IoT era as evidenced with verticals such as smart cities, smart transportation, smart health, smart homes, smart industries etc. [50], [94], [8] and [36].

The term IoT has been defined in multiple ways in the literature and various standardization bodies. Most of these definitions revolve around "smart objects". That is, objects with sensing and/or actuation capabilities. As per the Oxford dictionary definition, IoT is a "proposed development of the Internet, in which everyday objects have network connectivity, allowing them to send and receive

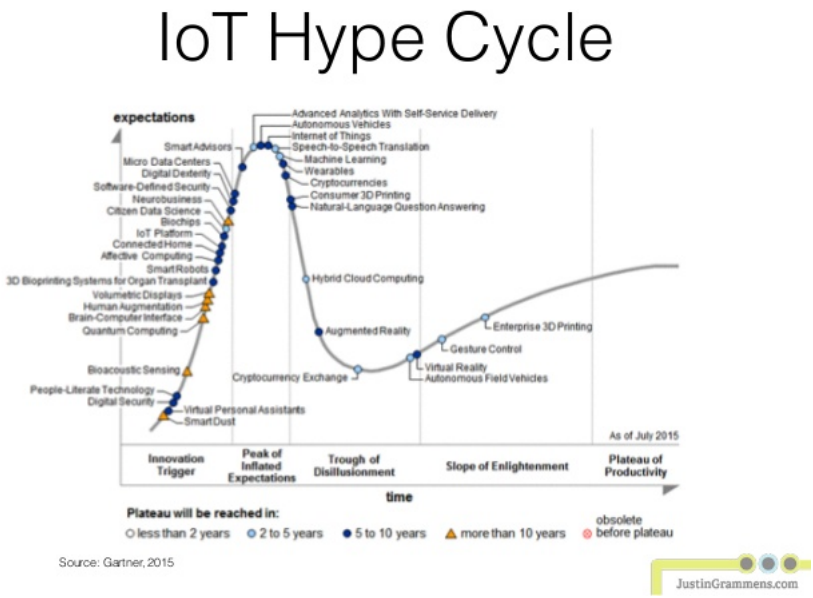

Figure 2.1 The IoT hype curve, according to Gardner.

data". ITU defines IoT as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" [10]. As a historical fact, shown in Figure 2.2, Cisco considers 2008 as the start of IoT era; this is when the number of devices per person became larger than the number of people. The study in [82] also predicts that the number of connected devices per person to increase from 1.6 in 2010 to 6.58 in 2020. In Ashton's [12] (the first person to coin the term IoT) words, the ability monitoring and controlling things/the physical environment over interoperable information and communication technologies has the potential to change the world, just as the Internet did.

The volume of data traffic is consistently increasing with the rapid adoption of IoT. Looking to the future, Cisco predicts there will be 50 billion devices connected to the Internet by 2020 [82]. Another estimate predicts that, by the year 2020, the Internet will connect nearly 32 billion 'things' generating approximately 44 Zettabytes of data [63]. The prevailing opinion today is that most of this IoT data will get processed and analyzed in *cloud datacenters* (as opposed to in-house processing facilities). This huge volume of data poses significant challenges to cloud service providers.

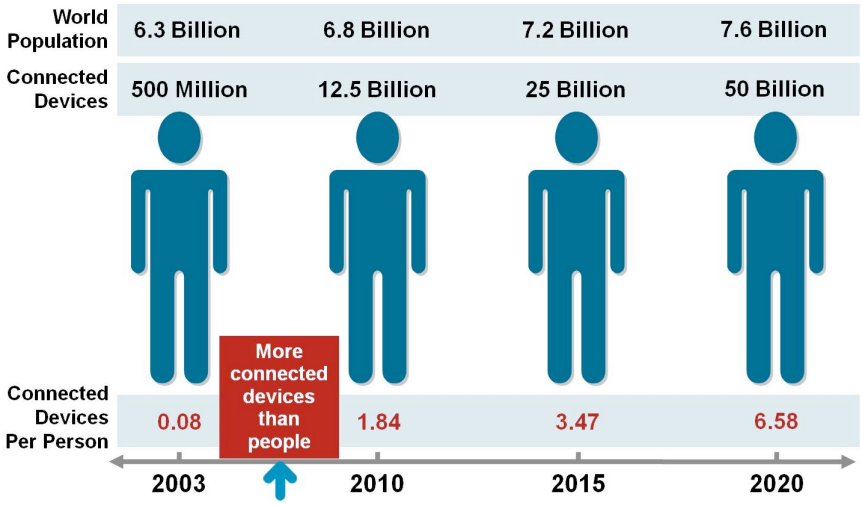

Figure 2.2 A history of IoT. Source [82].

2.2 IoT Cloud Computing

The computing model for IoT ranges from fully distributed (at the smart objects) to fully centralized (at the cloud datacenters) computing [26], [50] . Cloud computing has been used extensively in deployment and management of IoT applications in different IoT verticals like smart health [37], [52], smart home [32], smart cities [79], [69]. At present, big cloud vendors like Amazon, IBM, Microsoft, and Google are offering their IoT cloud PaaS solution to be part of current IoT evolution.

According to the NIST definition of cloud Platform as a Service (PaaS) [66], PaaS is *the capability provided to the consumer to deploy onto the cloud infrastructure consumer applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.* To an IoT user, the IoT cloud platform offers data collection, processing, analytics and visualization services while at the same time hides the complexity of managing all resources. Combining resource elasticity with flexible, “pay per use” pricing models make cloud computing a viable option for IoT users.

An IoT cloud platform allows tenant’s devices and applications to connect and securely interact with cloud applications and other devices. IoT clouds also offer other IoT specific services like device management, device and application security, quality of service, policy management and device & application monitoring, etc. In Figure 2.3, we show an example of IoT Cloud PaaS solution from

IBM, in which measurements and actuator controls are transported to/from IoT Systems under management (SUM) to IBM's IoT Platform using MQTT [68] as a transportation protocol.

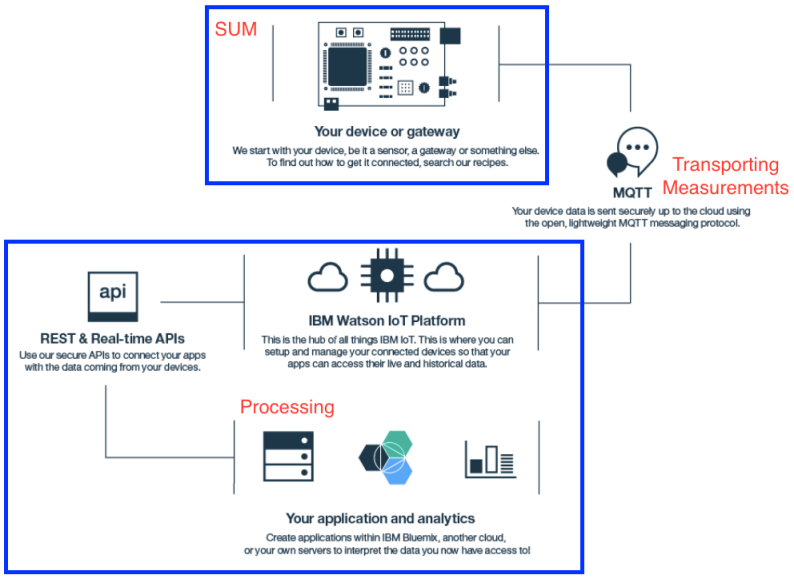

Figure 2.3 An example of IoT Platform as a Service (PaaS) cloud solution. Source [54].

2.2.1 IoT Cloud Services

In general, the key elements of a cloud IoT solution are the following.

Cloud Pub-sub service: the cloud pub-sub (aka message broker service) is the crux of IoT cloud deployments. The IoT message broker offers connectivity services to the devices and applications

in the form of a publish and subscribe (pub/sub) environment.

Secure registration for tenant's devices and applications: this service allows tenant's devices and applications to register with the IoT cloud. Access security key token allows these registered devices/applications to communicate using IoT cloud.

Device management: many cloud vendors are offering management for (a large number of) tenant's premise devices. This service includes provisioning and authentication, configuration and control, monitoring and diagnostics, and software updates & maintenance for devices in the IoT tenant premises, via the cloud.

Databases and analytics services: IoT sensor data can come in real or non-real time, temporal/spatial flavors; hence it requires various database services. For the same reasons, the analytics services cover the entire spectrum of monitoring, behavioral, and predictive analytics.

In this thesis, we focus on cloud pub-sub services only.

2.2.2 Service Requirements for IoT Cloud Computing

Users of a service place a number of requirements, that the IoT cloud provider must meet; typical such requirements includes (but are not limited to):

- **Security:** Security is a major challenge for IoT Cloud providers. It requires a secure connection from multiple tenant devices and applications to IoT cloud.
- **Scalability:** Guarantee on support for a large number of devices and a huge volume of data going in and out of the cloud, and scaling up or down to adapt to a potentially variable load.
- **Performance:** This requires cloud providers to make sure that the tenants get pre-specified (average or real) response times.
- **Availability:** A guarantee that the services and applications are (nearly) always available for access. A typical example is that the cloud applications and services should be available for access for 99.999% time over a period of one month.
- **Support for different traffic patterns:** Sensor data generated by smart things can result in different patterns when they arrive at the cloud datacenter. For instance, in an IoT cloud application to monitor city traffic, the patterns can be easily formed, both on temporal (rush hour, weekend/weekdays, summer/winter) and spatial dimensions (particularly hot spots, traffic signals etc).

- **Device Management**: This requirement pertains to providing provisioning and authentication, configuration and control, monitoring and diagnostics, and software updates and maintenance for IoT tenant premise devices via cloud.

2.2.3 IoT Cloud SLA

2.2.3.1 Generic Cloud SLA Applicable to IoT

The service requirements are typically communicated to the IoT tenants using Service Level Agreements (SLAs). These SLAs cover tenant and provider responsibilities regarding service requirements. Current SLAs offered by cloud providers focus primarily on the availability requirement; we se this for their IaaS, PaaS, as well as SaaS offerings [78], [54], [7]. A generic availability SLA may be stated as follows:

Availability SLA: The cloud applications and services will have a monthly uptime (i.e., they should be available for access) of 99.95%; here, *Monthly Uptime % = (Maximum Available Minutes-Downtime) / Maximum Available Minutes. In case of non-conformance, the provider will give the tenant service credits (usually 10% or 25% refunds if the uptime is less than 99.95% or 99% respectively) by adjusting next month's bill.*

Currently, we do not see many service-specific SLAs from cloud providers that specify a KPI other than the availability metric [47], [77].

2.2.3.2 IoT Specific Cloud SLA

Current cloud provider offerings focus on Pub/sub IoT services and offer IoT specific SLAs. For simplicity, the current, commercial, IoT-specific SLAs, such as Amazon's AWS [17], Microsoft's Azure [18], and IBM's Bluemix [53], *include only a single metric, namely volume of traffic*. In such SLAs, the volume of traffic is typically measured in terms of "messages" published to or delivered by the IoT broker service to IoT applications and devices. The broker service SLA offered does not take into account any other features, for example application traffic patterns. The cloud providers choose to ignore other features in order to simplify the (official, legal and binding) statement of the SLA.

We mention next a few examples of current, commercially available SLAs.

- *From Amazon AWS IoT:* Charges $5 per million messages processed, per month. A message is defined as a 512-byte block of data; processing is either published to or delivered by the IoT service [17].

- *Google IoT Cloud Pub-Sub:* This service has a tiered charging scheme. It charges $0.40/million for the first 250M operations and drops the rate down to $0.05/million beyond 1,750 Million operation. Here, an operation is to publish/subscribe a message (upto 64kB) to a topic [46].

- *Microsoft Azure IoT:* Tiered service with 400,000 messages/day for tier 1 to 300 million messages/day for tier 2 across all connected devices. The pricing varies from about $50/month to $5,000/month for tier 1 and tier 2 respectively [18].

- *IBM Watson IoT:* Tiered pricing where tier 1 allows 450GB over a month charged at $0.001 per MB exchanged, whereas the highest tier allows 7TB charged at $0.00014 per MB exchanged [53].

In the above SLAs, in addition to the volume metric, each cloud provider attaches the availability SLAs (for example, 99.99% of time the service would be available to access). Another important point to consider is the refund policies offered by a provider in the case of non-conformance of the availability SLA. For example, 25% Service credits refunds (which will be adjusted in next month credit) in the case of less than 99.99% reliability [78]. There is is no explicit mention of service credits in the case the volume metric is not achieved, from any of the cloud providers.

2.2.3.3 Analysis of Volume-based, IoT-Related Cloud SLAs

In summary, for simplicity, current commercial, IoT-specific SLAs focus only on the volume of traffic requirement. In such SLAs, the volume of traffic is typically measured in terms of messages published to or delivered by the IoT broker service to IoT applications and devices.

The major strengths of the SLA (from both provider and tenant perspectives) are its simplicity of statement and its observability. Tenants can measure the total number of messages using local applications in their premises or their applications in the cloud. Cloud providers can also measure the total number of messages either at access or service tier of the cloud.

The major weakness of the SLA (from the provider perspective) is the difficulty of its enforcement. The cloud provider either has no knowledge of messages arrivals or does not want to complicate the statement of the SLA with traffic arrival information. This makes datacenter resource provisioning a difficult problem, even when one limits attention only to servers. We elaborate on this difficulty in more detail in the next chapters.

From the tenant perspective, a weakness of the volume-based SLA is the large time period (1 month) over which the guarantee is offered and enforced. This certainly leaves out applications with processing requirements that need guarantees at smaller than one month intervals (e.g., sensors that generate bursty or weekly, daily, hourly, etc. traffic patterns). Another weakness is the lack of

any performance (e.g., average delay) metric; this may limit the type of applications that are suitable to run with such an SLA. For example, real or near-real time applications may not have acceptable performance.

There is a fundamental weakness too, at least from an academic perspective, as we discuss in more detail in Section 4.1.2.

2.3 Motivation for SLA-based Resource Allocation in IoT Cloud Environment

2.3.1 Need for IoT SLA-based Resource Allocation

In general, one can say that the current SLAs are "tilted" towards the cloud providers. First, any large conformance period (such as a month) favors cloud providers during peak usage scenarios. Second, the consequences of non-conformance to the provider can be characterized as mild; they result in a small, non-cash financial penalty (a credit for next month's usage). The provider can be "lax" in their design and operation of resource provisioning.

In order to tilt the SLA towards the tenant, one can think of a few options. For example, the consequences of non-conformance can be made more severe: what if the credit was 100%? 150%? in cash? Alternatively, metrics other than the message volume can be introduced.

In both cases, the provider would be incentivized to provision resources with more thought. We discuss specific ways to design such provisioning rules in Chapter 6, in which the provider attempts to provide "acceptable average delay" performance, in addition to guaranteeing a volume metric. In order to address the fundamental weakness of this SLA, the provider can provision resources in the spirit of the approaches we discuss in Chapters 4 and 5.

Abusing terminology a little, in the remainder of this thesis, we call all such provisioning methods "(IoT) SLA-based Resource Allocation".

2.3.2 How is volume-based IoT Cloud SLA Different from a Networking SLA?

The IoT traffic SLA-based resource allocation is fundamentally different from the well-studied network traffic SLA-based allocation problem. The difference comes from the following major points.

First, in classical network environments, the volume-based provisioning is done for a single connection. The connection has "point to point" characteristics. Whereas in an IoT cloud, the volume-based provisioning is done for the notion of a "tenant". A tenant's traffic can come from

multiple input points and can be served at more than one service host. Second, in the network world, the SLA requirement is precisely defined in terms of measurable resources like bandwidth and buffer space. The networking SLAs are standardized [64], [87]. In the present state of the IoT cloud world, the volume-based SLA is not as well defined or standardized; there are no average or peak rate, maximum burst size etc. parameters specified. Last, in the IoT cloud, the resources are in terms of CPU, storage, and switch fabric/server bandwidth, as opposed to only network bandwidth in classical networks.

2.3.3 Why is IoT SLA-based Resource Allocation a Challenging Problem?

Despite the simplicity of its statement, enforcement of the current IoT SLA is a difficult problem for cloud providers. The SLA does not specify traffic patterns; therefore, the tenant can present their volume of traffic at the beginning or the end of the specified time interval; or s/he can spread the traffic uniformly. Allocating resources to "cover all possibilities" is the main challenge.

2.4 Related Work in SLA-based Resource Allocation

The traffic volume-based SLAs have been well studied in both classical IP networks and different cloud datacenter environments. In this section, we present related work in both environments. In each case, we also discuss how our work is different. We defer surveying work in the specific, narrow area of server scaling until Section 6.2, page 61, after we discuss server scaling in some more detail.

2.4.1 SLA-based Resource Allocation in Classical Networks

Overall, in Internet Service Provider (ISP) networks, there are two major types of SLAs: 1) Internet service SLA [13] (covers service requirements like availability, network latency, throughput, and packet loss etc.) and 2) VPN based SLA [67], [75] (covers requirements based on number of sites connected to the VPN, bandwidth, mode of connectivity, reliability, availability etc.). In the past, researchers and industry have put considerable effort to standardize the metrics used in the SLA (for example delay, latency, and data delivery ratio) [30], [91], [76]. In RFC 2205, RFC 2211 and RFC 2212, the authors provide specification of the end to end behavior (RSVP), the traffic (Tspec) and the desired service (RSpec) at the network element [30], [76]. TSpec standardizes traffic parameters like rate, bucket size, peak rate, minimum policed unit, maximum policed unit etc. In an IP network, the guaranteed service is invoked by specifying the traffic (Tspec) and the desired service (RSpec) to the network elements. In case of cloud IoT SLA, traffic or service specifications analogous to (Tspec)

and (RSpec) are not defined, let alone standardized. Therefore, resource allocation becomes quite challenging with different user traffic patterns and service needs.

Service level agreements in IP networks have also been studied in the past [64], [87]. In [87], the author gave a list of three environments (network service provider, hosted service provider, and an enterprise network) in network connectivity service and summarized three common approaches (insurance based, provisioning, and adaptive) to satisfy SLAs in IP network. Martin et al [64] studied SLAs modeled based on Frame relay/ATM/MPLS types of networks. They argued about the need for end-to-end SLA response time inclusion in the metrics covered in the SLA.

In IP networks, the SLA-based resource allocation dealt with the "connection notion"- for example from one ISP's router to another ISP's router or virtual circuit start/end point. In order to support the different requirements of tenants, network service providers use techniques like differentiated services [23] on network elements or traffic engineering using multiprotocol label switching (MPLS) [28], [15] on virtual circuits. To provide performance guarantees, researchers have published work in the area of scheduling (e.g. leaky bucket, WFQ etc.) [48], buffer management (e.g., RED), service definition and associated requirements (like controlled load service [76] and guaranteed service [91]).

Overall, our problem differs from classical traffic based resource allocation in network environments in three major points: (a) no traffic or service specification are present in the cloud service environment, (b) the cloud has traffic coming from multiple input points to one or more servers, and, (c) in cloud datacenters, the business model revolves around server resources (CPU, bandwidth, and storage etc.) as opposed to link bandwidth in IP networks.

2.4.2 SLA-based Resource Allocation in Cloud Computing

As we mentioned before, cloud providers currently offer a generic availability SLA for their IaaS, PaaS, and SaaS offerings, without considering the service nature and QoS requirements of tenant workloads. Today, more tenants are adopting cloud for different workload, and they are looking at more meaningful SLAs related to QoS requirements of their workload.

The major focus of research efforts has been defining meaningful SLAs for specific cloud service environments and providing a mechanism to cloud providers to enforce it. Traditionally, over-provisioning of resources to meet worst case demand is used in clouds with different workloads and service requirements. Over-provisioning of resources comes at a huge cost to the cloud provider. As a result, SLA-based resource allocation [31], in cloud datacenters has gained a considerable research interest over the past few years. Generally, these works differ based on the workload and cloud environment assumptions they make.

The authors of [92], [43], [84] have considered the SLA-oriented resource allocation through virtualization of physical server resources as virtual machines (VMs). Both Wu et al. [92] and Garg et al [43] have used heterogeneity of VM nature to map tenants with different QoS parameters (response time, service initiation time etc.) to appropriate VMs. Van et al. [84] consider average response time as the KPI; their objective is to keep it below a prescribed limit. Their control is (multiple variants of) VM placement. Our work is different from this line of research as we considered a different KPI in the SLA and use different controls to achieve it.

The authors of [42] and [93] have considered heuristic approaches for SLA-based resource allocation. Fu et al [42] proposed an SLA-based dynamic scheduling algorithm of distributed resources (bandwidth) for streaming workloads. Moreover, Yarmolenko et al [93] evaluated various SLA-based scheduling heuristics on parallel computing resources using resource (number of CPU nodes) utilization and income as evaluation metrics. Both of these approaches are different from our work due to workload and profit metric assumptions.

Researchers have also considered SLA-based resource allocation including average response time, number of jobs completed per unit time, tier pricing along with optimizing cost and profit for the providers. Boloor et al. [25] proposed a dynamic scheduling mechanism for response time SLA in SOA-based, context-aware applications. The authors of [24], [57] have proposed rate limiting the service requests to avoid overwhelming the service tier resources under an SLA bound. Our work differs from the above work in two ways: first, this work deals with delay percentiles, not averages, and second, our objective is different from protecting the server resources. The authors of [60] have used a dynamic scheduling algorithm than can guarantee SLAs related to CPU service share in clusters of shared server environments. In our work, the SLA is not expressed in terms of percentile CPU.

2.4.3 SLA-based Resource Allocation in IoT Cloud Computing

Past researches specific to IoT cloud have considered allocation strategies related to execution time and security groups. For example, Choi et al. [34] proposed a combinatorial auction system to efficiently allocate resources and reduce the penalty with execution time constraints. Vimercati et al. [88] proposed a framework for supporting application requirements related to security in a cloud for IoT data processing. In our work, we do not consider execution time or security groups. To the best of our knowledge, no work related to resource allocation for current volume-based SLAs is publicly available.

2.4.4 Differences in the Various Allocations

In the remainder of the thesis, we will consider two IoT cloud SLAs - we state them precisely in Sections 4.1.2, page 36, and 5.2.2, page 48. In summary, the resource provisioning problems that arise from the need to satisfy these SLAs in the cloud environment are different from similarly stated problems in the networking space in two main regards.

- In the cloud environment, the traffic is *multipoint-to-multipoint*, unlike the classical networking space.
- In the classical the networking space, unlike the cloud environment, SLA requirements are *precisely defined in terms of measurable resources* (like bandwidth and buffer space).

CHAPTER

3

SYSTEM MODEL

The chapter is organized as follows. In Section 3.1, we present the major elements of the system model. In the next section, we present two common models for deploying services in the cloud. in Section 3.3, we discuss the traffic patterns generated by IoT applications inside the datacenter. We outline the resources one can use for serving these patterns in Section 3.4. Finally, we list the available controls a designer can include in provisioning of those resources in Section 3.5.

3.1 Elements of the Physical System and System Model

An IoT cloud platform allows tenant's devices and applications to connect and securely interact with cloud applications, cloud services and other devices. To study our resource allocation problem, we use an abstract model of an end-to-end actual system with both tenant and cloud premises.

The major components of our system model are given below.

Tenant premises: the major components at the tenant premises are the following.

Sensors and actuators. Sensor devices collect measurements from the physical environment and actuator devices control actions performed on the physical environment. In our abstract model, sensors are presented by a set of measurements transported to the datacenter. Actuators are abstracted by a similar pattern. Both are further abstracted as the "messages" mentioned in the SLA

statements (for example, the SLA in Section 4.1.2, page 36).

IoT gateway. An IoT gateway allows devices at the tenant premises to communicate with the cloud. Moreover, some IoT gateways offer local processing and storage capabilities. In our abstract model, gateways generate the various *traffic patterns of measurements* as we detail in Section 4.3.2, for example

Tenant applications. Generally, a tenant deploys applications in both datacenter and the tenant premises. This element does not affect our model.

Cloud datacenter: the major components at the datacenter are the following.

Access gateways. These are the end-point routers which provide access to the cloud applications and services. This element does not affect our model.

Switch fabric. This network connects multiple servers to the access gateways. The topology in which the switches/routers of the switch fabric are connected (e.g., hierarchical, fat tree, leaf-spine etc. [62]) is important, as it defines the bandwidth resource. This element does not affect our model, since we do not consider provisioning of bandwidth in this thesis.

IoT message brokers. An IoT message broker is a pub/sub broker service running on a physical host server machine. The broker enables devices and applications to send to or receive messages from other devices and applications. When communicating with IoT cloud, a client publishes a message addressed with a topic name to message broker. The message broker, in turn, sends the message to all clients that have subscribed to messages for that topic.

The service can be hosted in a virtualized environment, running as a container or VM and sharing the CPU/memory resource of the server with other applications; it can be running in a physical server by itself. In our model, we abstract the IoT message broker service by considering only its *capacity to process the (abstract) messages.*

Cloud applications and services. Cloud applications allow device management and other services orchestration. The cloud services (e.g. database, analytics, etc.) help to build IoT applications that gather, process and analyze the data generated by connected devices and applications. This element does not affect our model.

The variety in tenant applications and sensor/actuator capabilities present a wide scale of requirements to the cloud provider in terms of scale, performance, security, isolation etc. The cloud provider offers different deployment models to organize the infrastructure (i.e., access gateway, switch fabric, service instances) to meet this spectrum of tenant requirements. In next section, we discuss a generic cloud deployment model with two variants to deploy IoT related applications in the cloud. We focus on the server resource only.

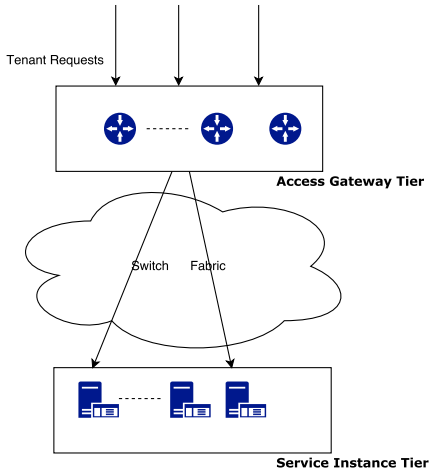

Figure 3.1 Single tier service cloud deployment model.

3.2 Cloud Deployment Models

Cloud deployment depends upon the services offered. The service tier (that is the collection of all servers) in a cloud can be deployed as single or multitier architecture. In both, the service can be deployed in a single or multiple instance for fault-tolerance, performance, and security reasons.

3.2.1 Single-tier Service Cloud Deployment

As shown in Figure 3.1, in single tier service instances, traffic from access gateway can be sent directly to the service tier. The single tier of service instance is a simple cloud deployment but offers limited control to provider.

Cloud provider applies the control at the access gateway tier or at the service tier to satisfy the service level agreement or performance guarantee for both tenant and provider infrastructure. In most deployments, access gateways are the routers at the edge of cloud networks and their main task is to distribute the service load coming from a tenant among the service instances without any deep packet inspection. Service instances in cloud are also deployed in multiple ways, offering varying degrees of control and constraints to satisfy the service level agreements. A service instance per tenant offers simplicity in terms of isolation, but limits performance. A cluster of service instances per tenant offers performance but presents additional challenges in terms of instance selection to serve a request. A shared server improves efficiency, but presents a challenge in terms of isolation and scheduling.

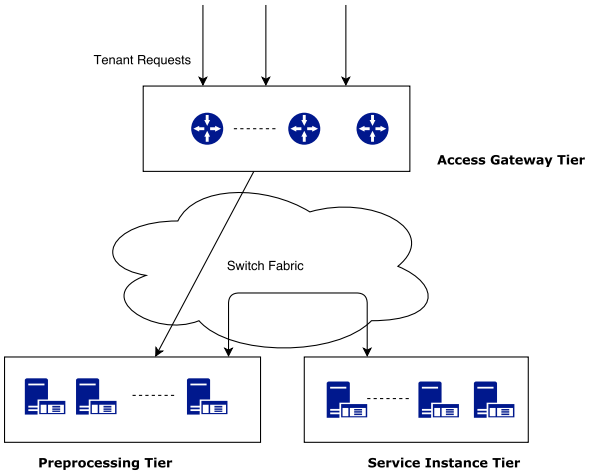

Figure 3.2 Multitier service cloud deployment model. Preprocessing and service tier are two tier of service instances.

3.2.2 Multitier Service Cloud Deployment

As shown in Figure 3.2, in multitier architecture the access gateway tier send traffic to the pre-processing tier in order to get access to service tier. A multitier architecture allows provider to have more granular control over the traffic and presents additional control to satisfy the service level agreement or performance guarantee for both tenant and provider infrastructure.

A pre-processing tier of devices can be deployed as single or cluster instances, offering functionalities like application proxy, service integration, intelligent routing, application data pre-processing. For example, a pre-processing device protects service instances by storing sudden surge in incoming traffic and applying shaping mechanism for traffic towards service instance. Another example of prepossessing tier functionality can be web proxy, terminating HTTP connections and doing intelligent load balancing on back-end web application instances.

Deployment of a multitier model is complex, in comparison to the single tier model. The multitier models introduces additional EW (East West) traffic to a datacenter. Depending upon number of pre-processing and service instances there can be different model of EW traffic scenario: (a) point to point (a pre-processing instance to multiple service instance) (b) point to multipoint (a pre-processing instance to multiple service instance) (b) multipoint to multipoint (multiple pre-processing instance to multiple service instance). The instances in preprocessing and service tier can be single, shared or clustered as per tenant and provider requirement.

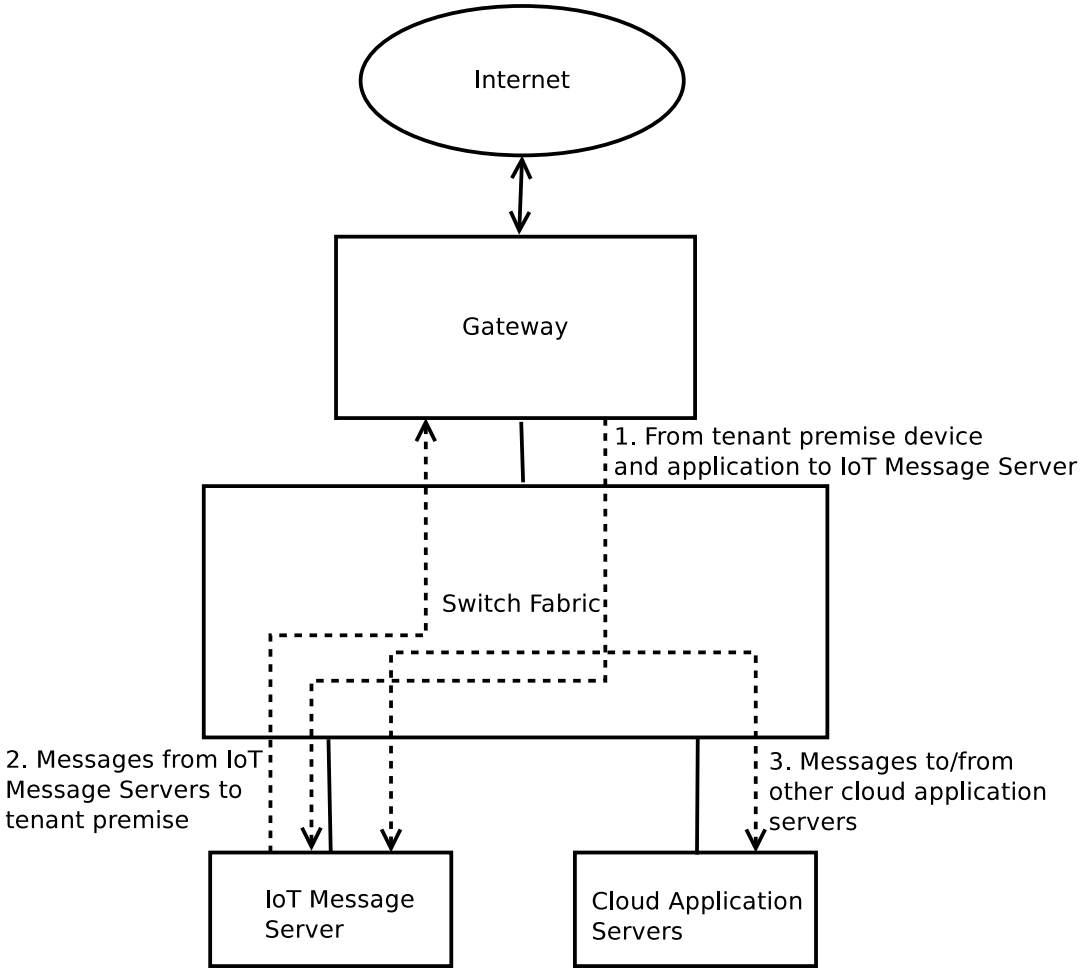

Figure 3.3 Traffic flows in a IoT cloud datacenter.

3.3 IoT Applications in the IoT Datacenter

The traffic patterns for a generic pub/sub IoT application are shown in figure 3.3. The key traffic flows are:

- *Gateway tier to IoT servers/preprocessing server:* Messages published by tenant devices and applications to IoT broker service. This generates a NS (North-South) pattern in IoT cloud.
- *IoT server/ preprocessing server to gateway tier:* Messages subscribed by tenant devices and applications from IoT broker service. This generates a SN (South-North) pattern in IoT cloud.

- *Preprocessing server to IoT server:* Preprocessed messages published by tenant devices and applications to IoT broker service. One preprocessing example can be storage tier to store the traffic before forwarding it to an IoT server. This generates EW traffic pattern in the datacenter.
- *IoT server to cloud services server:* Messages published or subscribed by cloud applications and services from IoT broker service. This generates an EW traffic pattern in the datacenter.

3.4 Resources in the IoT Datacenter

We define resources as the component of cloud datacenter for which there is contention among the tenants. A provider needs to do intelligent resource allocation to meet the service requirement and maintain profit too. The current IoT specific SLA is expressed in terms of volume of messages processed in cloud datacenter, so we consider only server resource (to process the messages) in the thesis. In the literature, several units have been used to quantify the amount of resources available: number of servers, number of VMs, CPU cycles are the most common; main memory has also been used, but less frequently.

3.5 Controls in the IoT Datacenter

Dividing the server resource among competing tenants can be done by using several controls (aka "knobs"). The most commonly used are listed below.

- Server CPU scheduling - it provides a mechanism to share CPU cycles among tenant applications.
- Rate limiting tenant traffic - it provides an upper limit on the amount of traffic a tenant can send to the datacenter. Usually it is associated with dropping excess traffic.
- Traffic shaping - it provides a similar upper limit, but stores excess traffic instead of dropping it.
- Server load balancing - in the case of multiple service instances, load balancing can increase the "performance" of system.
- System parameters and configurations - Total server capacity, number of servers, service instance placement.

In our designs, we will use the first, second and fifth knobs.

CHAPTER

4

RESOURCE ALLOCATION FOR CURRENT IOT SLA - SINGLE SERVICE INSTANCE

As we discussed in Section 2.2.3.3, page 23, the way current commercial, IoT-specific SLAs are structured represents a tradeoff that leans towards simplicity of statement. In Section 4.1, we present the problem formulation. We formally state the SLA using the model of the previous chapter. We argue in Section 4.1.2 that the SLA *cannot actually be met*, via any control the provider's designer can devise. This is due to the lack of traffic specifications. In Section 4.2, we explain how we address this issue and provide an ameliorating solution to conform to the current IoT SLA. In Section 4.3, we describe the experiments conducted and results obtained. Finally, in Section 4.4, we conclude the chapter.

4.1 Problem Formulation

4.1.1 System Architecture

The topology considered for this chapter is given in Figure 4.1. In our system, we consider a single message server shared by multiple tenants. Multiple gateways in the access gateway tier forwards

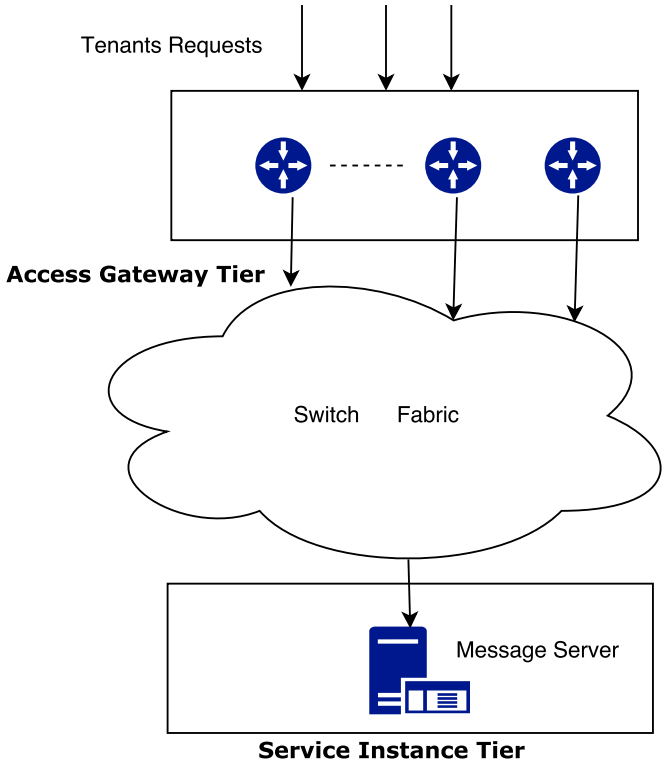

Figure 4.1 Topology considered: Single service instance (message server) for all tenants.

traffic received from multiple tenants to this single message server. The detailed description of system elements in this topology is already given in section 3.1, on 29.

4.1.2 Formal IoT SLA

In volume-based IoT SLAs, the volume of traffic is typically measured in terms of messages published to or delivered by the IoT broker service to IoT applications and devices. In the spirit of Amazon's AWS [17], Microsoft's Azure [18], Google Cloud Pub-Sub [46], and IBM's Watson [53] volume-based SLA, we state the SLA as follows:

(SLA1) "*For a fee of $Y to be paid by the tenant, the provider will supply an IoT message brokering service that will process a total of X IoT publish or subscribe messages over a time period T.*"

Generally, an IoT message is a 512-byte block of data processed by the IoT Service. Typical value for T in [17], [18], and [53] is one month.

When taken literally, the promise to process a total of X messages in T units of time is just that - a promise that cannot be *guaranteed unconditionally*. At least not without conditions on how messages are sent to the datacenter. Consider, for the sake of a simple counterexample, the case of a (potentially malfunctioning) tenant gateway that bursts all of its gathered sensor measurements arbitrarily close to the end of the time period T. All X_i messages from a tenant i may arrive close to T. The amount of CPU resource the provider would need to provide cannot be bounded. In addition to this (fatal, if one talks strictly academically) flaw, what happens if tenant i sends more (less) than X_i messages over the time period T? There is no description of what to do in these situations, again for the sake of simplicity.

In this chapter, we retain the statement of the SLA. Recognizing the fact that it cannot be guaranteed, we proceed to design a control that ameliorates the problem.

4.1.3 Definition of Key Terms

We define some key terms used in the formulation of our objectives as follows.

Definition 1. **Measurement Period** (T): this is the time interval, specified in the SLA statement, that is used to measure the number of message arrivals. △

Note that this value is universally applied to all tenants, in each one of the cited, commercial SLAs.

Definition 2. **Number of Tenants** (M): we consider an IoT cloud with M tenants; each tenant i sends X_i messages to the IoT broker service over a time T. To avoid unnecessary complications in restating (SLA1) (as well as to limit the number of experiments in the evaluation section), we assume that tenant i sends exactly X_i messages. △

For notational simplicity, we assume that each message requires 1 unit of time to be processed. Therefore, we need a minimum server capacity of X_i/T messages/second (to have a chance) to meet the SLA for a given tenant i.

Definition 3. **Message processing capacity** (C): C is the total message process capacity in the datacenter. △

To meet the requirements of all tenants, this capacity must be overprovisioned, beyond the required minimum $\sum_i X_i/T$ messages/second bound. We will quantify by how much soon, in Section 4.2.

Definition 4. **Enforcement Period** (EP): this is the time interval needed to *unconditionally* guarantee that the SLA is met. △

We have:

$$EP = T + \Delta t, \tag{4.1}$$

where Δt, discussed in Section 4.2.2, is a control parameter at the disposal of the designer. It is introduced so that the provider can guarantee that X_i messages will be processed by time EP (instead of T). Formally, the value of Δt could depend on the tenant; for notational simplicity, since T is assumed the same for all tenants, we ignore this dependence.

4.1.4 Objectives

Consider an IoT cloud with M tenants; each tenant i, where $i = 1, \ldots, M$, sends (upto) X_i messages to the IoT broker service over a time period T. We state our resource allocation objective as follows:

(O1) "For tenant i, provide service resources to meet the SLA by the enforcement time period ($T + \Delta t$)."

4.2 Our Approach

4.2.1 Control Parameters and Constraints for Achieving SLA

In this subsection, we discuss the control parameters available to the cloud provider to meet objective (O1). As mentioned in Section 3.5, page 34, among the generally available controls, we choose to utilize:

- Server CPU scheduling;
- Rate limiting tenant traffic;
- System parameters and configurations. In this chapter, these include total server capacity and Δt, the duration of the additional enforcement period, as explained in Figure 4.2.

Constraints. The cloud provider has the following two major constraints to achieve the SLA.

- No knowledge of arrival patterns.
- No control over excess message arrival from a tenant: there is no mention in the SLA about how to handle excess traffic.

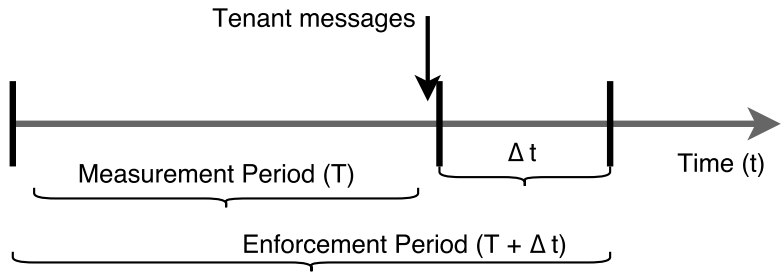

Figure 4.2 Total enforcement period.

4.2.2 Design Considerations

4.2.2.1 Choosing the Additional Enforcement Time

As shown in Figure 4.2, if all the X_i messages from a tenant i come arbitrarily close to the end of the enforcement period T, conforming to SLA would require infinite resources. In order to handle this case, without the need to complicate the statement of the SLA with traffic arrival information, we propose to extend the enforcement period to process messages of the previous measurement period by an additional time, Δt. The provider has $T + \Delta t$ as the enforcement time to guarantee processing of the messages arrived in time T. Suppose in the worst case scenario, all X_i messages from a tenant i come at the end of period T, the provider can use an additional Δt time to process these X_i messages. This Δt time depends, of course, on the processing capacity at the server tier, the number of tenants, and number of messages per tenant in the enforcement period etc. For simplicity of notation, we have dropped the dependence of Δt on i.

The value of Δt is determined by how much we overprovision server capacity. Suppose, for example, that we choose

$$C_i = nX_i/T$$

Then, in the worst case scenario of all messages arriving at time T, a server with capacity C_i will finish processing at time $T + T/n$. Hence

$$\Delta t = T/n, \tag{4.2}$$

where n is the overprovisioning factor.

4.2.3 Assumptions

No assumptions in addition to the ones made in the model formulation of the previous chapter are made in this chapter.

4.2.4 Algorithm to Achieve the Current SLA

4.2.4.1 Discussion of Variables

In our algorithm, we consider total number of tenants as (M). We run the algorithm for an indefinite number of enforcement periods of duration T each. As per current SLA, for each tenant i, the provider needs to provide (overprovisioned) server resources to process X_i messages sent over a time T. We fixed the maximum allowed messages for tenant i over time T to M_i.

The number of messages arrived and messages processed for tenant i during the n_{th} enforcement period (where $n = 1, 2, 3, \dots$) are denoted as A_{in} and P_{in} respectively. D_{in} denotes the minimum number of requests from the n_{th} enforcement period to be processed in the $(n+1)_{th}$ enforcement period.

4.2.4.2 Algorithm

We propose to incorporate the following control knobs in the resource allocation mechanism to meet objective (O1).

- *Overprovisioning factor n*. As seen from Equation 4.2, Δt is dependent on server capacity. The overprovisioning factor determines how "fast the guarantee can be met", even in the worst possible case.

- *CPU scheduling*. As tenant arrivals are independent of each other, we propose per tenant CPU queuing. We denote the set of queues as Q in the description of our algorithm. We propose a two-tier scheduling system, with (logically separate) "regular" and "overflow" queues. Regular queues are served in weighted round robin manner with a weight (w_i) of tenant i being $X_i / \sum_{j=1}^{M} X_j$. If a tenant i has sent more that X_i number of messages during the enforcement period, the excess traffic is placed in the overflow queues. Regular queues get scheduling priority over overflow queues.

- *Policing*. We propose to exercise this control to limit the input rate of the messages. (Note that with the presence of overflow queues, policing is not strictly required. We mention it here simply as an alternative to simplify the scheduling control.)

The mechanism to provably achieve the current SLA (within the extension period) is described in Algorithm 1.

Algorithm 1 Over-provisioning of server capacity, WRR scheduling and rate limiting for tenant traffic

Output: Indication that the SLA was met or not, for each tenant.

```
for all enforcement periods n, where n = (1,2,3,...) do
  Set D_i0 = 0, ∀ i ∈ {1,...,M}.
  on every arrival, allow a message from a tenant i only if A_in < M_i and set A_in = A_in + 1
  while CPU is free and ∃ a non empty queue in Q do
    if ∃ a set S ⊆ Q where P_in < X_i + D_i(n−1) then
      Run WRR scheduler for queues in S
      If tenant i is selected then P_in = P_in + 1
    else
      Run WRR scheduler for queues in Q
      If tenant i is selected then P_in = P_in + 1
    end if
  end while
  At the end of the n_th enforcement period
  ∀ i ∈ {1,...,M}, set D_in = max{0, min[(X_i + D_i(n−1) − P_in), (A_in + D_i(n−1) − P_in)]}
  At the end of the additional enforcement time of n-th enforcement period,
  for all tenants i ∈ {1,...,M} do
    if D_in = 0 then
      SLA met for tenant i
    else
      SLA not met for tenant i
    end if
  end for
end for
```

4.2.4.3 Why such a design?

From the previous discussions, it should be clear that the provider supplies plenty of server capacity to serve all tenants. Why isn't a simple, common buffering mechanism along with FIFO scheduling sufficient? The answer is that this scheme would work only if tenant i sent up to X_i messages during time interval T. Without this condition, separate buffers are needed, to isolate tenants. Then a scheduling policy other than FIFO must be designed.

4.2.4.4 Proof that the algorithm works.

We defer the proof till the next chapter. The notion of enforcement periods here is analogous to the notion of subperiods in Section 5.3.4. The with the obvious changes in notation, the reasoning is exactly the same.

4.3 Experimental Evaluation

4.3.1 Experimental Setup

The simulation of broker service in IoT cloud datacenter is implemented as a single threaded discrete event simulator in C. The simulator is implemented with data and model separation. For simplicity, the time to process a message is modeled as a constant. The simulator permits changing the arrival rate, service capacity, and duration of measurement and the enforcement period. Unless otherwise specified, we set T to one month and the total number of messages arrival to one million.

4.3.2 Traffic Patterns

We generate the following traffic patterns by defining the instantaneous arrival rate, $AR(t)$, as indicated below, for all $t \in [0, T]$.

- **Worst case traffic.** The sensors burst all their measurements at the end of the measurement period T in a burst of duration b. Therefore,

$$AR(t) = \begin{cases} X/b, & \text{for} \quad T - b \leq t \leq T \\ 0, & \text{otherwise.} \end{cases}$$

  We select duration of burst as *burst size/bandwidth*, where bandwidth is fixed to 1 Mbps throughout the simulation.

- **Uniform Traffic.** We assume that the sensors send their measurements uniformly over the time interval $[0, T]$. Therefore,

$$AR(t) = X/T, \quad \text{for } 0 \leq t \leq T.$$

- **Excess traffic** We have assumed a special case of uniform traffic, where a tenant's sensor sends X_1 ($X_1 > X$) measurement. Therefore,

$$AR(t) = X_1/T, \quad \text{for } 0 \leq t \leq T.$$

4.3.3 Evaluation of Proposed Solution

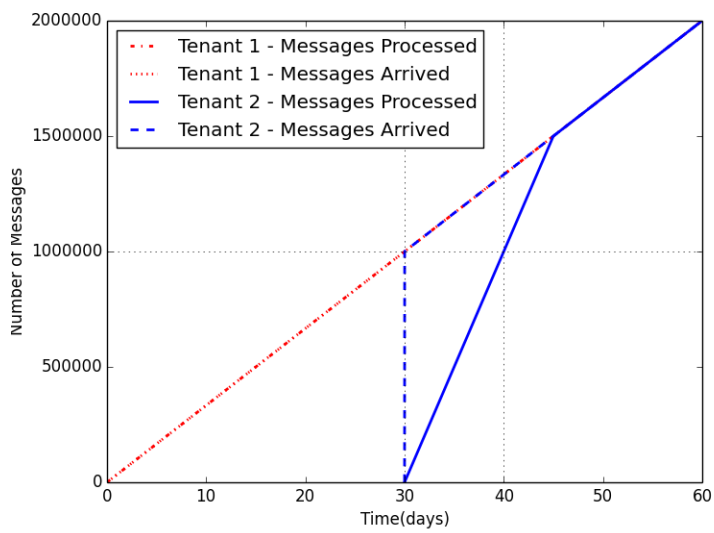

Figure 4.3 SLA Conformance.

We center the evaluation of our algorithm around three questions.

In order to only visualize results, since the algorithm is proven to satisfy the SLA, in the first question we **"verify that Algorithm 1 satisfies the SLA"**. For this, we run an experiment with two tenants, with values $X_1 = X_2 = 1M$ messages, $T = 1$ month and overprovisioning factor $n = 2$ (for each tenant). Tenant 1 sends messages uniformly throughout T and tenant 2 sends all his/her messages at the end of the measurement period T. As shown in Figure 4.3, the number of processed messages for tenant 1 is (practically) 1M by time T, and thus the SLA is satisfied. For tenant 2, the guarantee is to finish processing of 1M messages by time $T + T/2$, that is 45 days. This actually happens in less time (around day 40 in the figure) because the excess capacity for tenant 1 is not needed and thus is used for processing messages that belong to tenant 2.

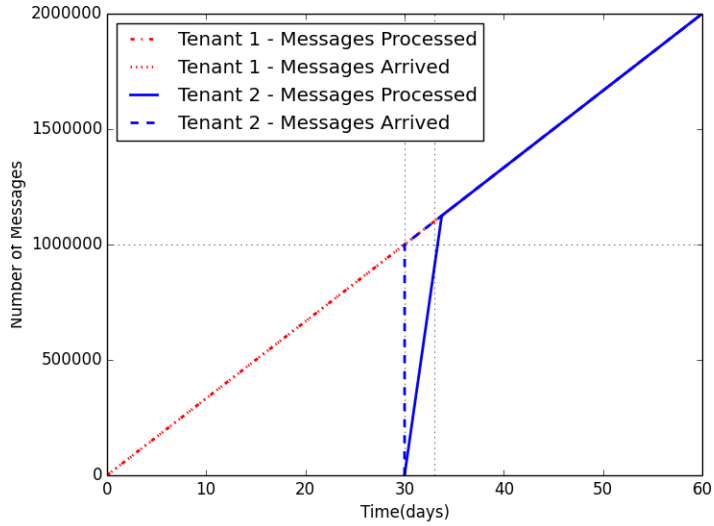

Figure 4.4 The effect of service capacity overprovisioning.

We then proceed to verify that **"by over-provisioning service capacity by a factor of n reduces the additional enforcement period to T/n"**. For this, we run an experiment with two tenants, with values $X_1 = X_2 = 1M$ messages, $T = 1$ month and overprovisioning factor $n = 5$ (for each tenant). From Equation 4.2, page 39, all messages for tenant 2 should be processed within[1] $T/(2n-1) = 30/9$, approximately 2.7 days after the measurement period. Now, with additional processing capacity, the excess pending requests of previous measurement period are processed a lot earlier as shown in Figure 4.4.

The third question we answer is, **"does the algorithm work correctly in the case of excess traffic sent to the cloud by the tenant"**? Algorithm 1 uses "overflow" queues that are supposed to be scheduled for service only when there is CPU time available. (Note also that the algorithm limits arrivals to M_i, a configured threshold.) To verify correct operation, we run an experiment for two measurement periods ($T = 1$ month) with two tenants and overprovisioning factor $n = 2$ (for each tenant). We set $X_1 = 1M$ and $X_2 = 1M$ messages and use $M_i = 2M$ for both tenants; this value allows $2M$ messages per measurement period T. Tenant 1 sends the maximum allowed $4M$ messages uniformly throughout $(0, 2T)$ - thus in excess of the SLA; tenant 2 sends $X_2 = 1M$ messages in a burst at the end of the measurement period T and then sends $2M$ messages uniformly throughout

[1] Note that in calculating the reduction in the additional enforcement period, when we apply Equation 4.2, we must use the *total* excess capacity. With two tenants, this is $2n-1$. The term -1 accounts for the fact that tenant 1 traffic is already using the server.

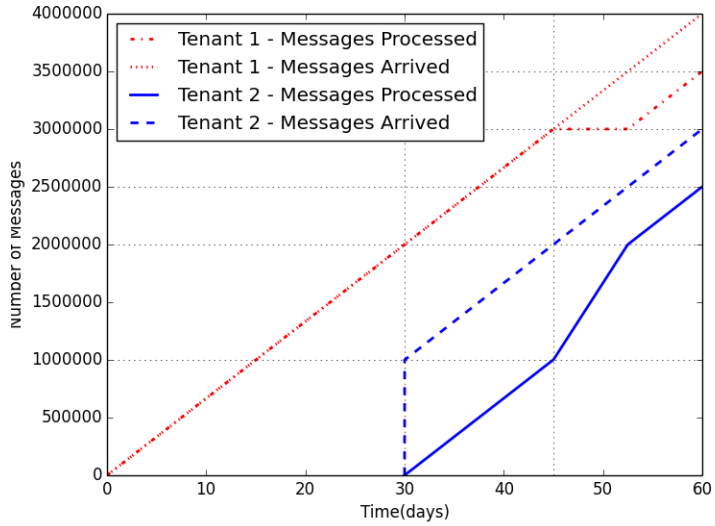

Figure 4.5 SLA Conformance in case of excess traffic.

$(T, 2T)$.

Operation during $[0, T]$ is as expected. As we see in Figure 4.5, all of tenant 1 messages get processed; that includes the $1M$ excess messages. As far as tenant 2 goes, the $X_2 = 1M$ messages that arrived at T are processed by day 45, which is the guaranteed additional enforcement period ($T/2 = 15$ days). During $(T, 2T]$, we see the effect of the proposed regular and overflow queues. At day 45, tenant 1 has already processed the $1M$ messages that arrived in $(T, 1.5T]$, so it's remaining messages go into the overflow queue. Whereas on day 45, tenant 2 has just finished processing messages from the previous measurement period and the messages arrived in $(T, 1.5T]$ are in the regular queue. At day 45, message scheduling stops for tenant 1 until $1M$ messages for tenant 2 are processed; this takes an additional $T/4 = 7.5$ days. At day $45 + 7.5$, both tenants resume service simultaneously from the overflow queues. As designed, the excess traffic from tenant 1 in the overflow queue does not affect the performance of the other tenant.

4.3.4 Overhead and Tradeoff Analysis

In the case of a large number of tenants, per tenant queues at the message broker server increases overhead. We can limit this overhead by adding more message broker servers and implementing server load balancing mechanism. Since queues within the SLA limit get priority over queues with excess traffic, the variance in response time can be huge. To limit variations in message response

time, a dynamic weight scheduler can be implemented.

4.4 Conclusion

In this part, we studied conformance of commercially offered, volume-based, IoT SLAs as a resource allocation problem. We proposed a buffering, scheduling and rate limiting mechanism to provably enforce these SLAs. Our evaluation shows that our solution achieves conformance within a fixed additional enforcement time period without the need to specify information about arrival patterns and works even in the case of excess traffic. We also showed that the increase in the service capacity and decrease in the measurement time both reduce the length of the additional enforcement period to conform to the SLA.

CHAPTER

5

RESOURCE ALLOCATION FOR A NOVEL IOT SLA - SINGLE SERVICE INSTANCE

The chapter is organized as follows. In Section 5.1, we discuss the reasons that call for a new SLA statement. In Section 5.2, we formally state the SLA and formulate the new problem. We discuss a new control and define a solution that uses it in Section 5.3. In Section 5.4, we describe the experiments conducted and results obtained.

5.1 Need for New IoT SLA

As discussed (ad nauseam) already, the strict enforcement of the current, volume-based IoT SLA is a challenging, technically ill-defined problem. In the previous chapter, we proposed a "patch" (the additional enforcement time as a parameter) needed to conform to the SLA in fixed time. We fix the total number of arrivals in an enforcement period and work on to reduce the additional enforcement time to process the promised number of messages. From Equation 4.2, once we reduce the enforcement time, the additional enforcement time also reduces. Intuitively, by reducing enforcement time we are putting a rate limiter on message arrival by a tenant (maximum number of messages promised to be processed in an enforcement period). It helps providers to deal with the

constraint of no knowledge regarding arrival patterns of a tenant's traffic. Based on this idea, we propose in this chapter a new SLA with inherent rate limiting feature.

Overall, we believe that it makes sense for cloud providers to consider a new SLA which includes the following characteristics: 1) retains the simplicity and observability of the current SLA, 2) offers additional control "knobs" for enforcement, and, 3) provides more granular control over enforcement period as some IoT tenants may have time sensitive measurement and control data.

In this chapter, we propose such a SLA. In particular, we propose a two-step SLA introducing concepts of subperiods of an enforcement period. We also discuss major strengths and weaknesses of the proposed SLA. We formulate the enforcement of the new SLA as a resource allocation problem and propose an algorithm to solve it. We prove theoretically that the algorithm will guarantee the SLA. Our experiments show that the concept of subintervals simplifies the enforcement of the SLA for the providers. It also improves the average delay performance for tenants.

5.2 Problem Formulation

5.2.1 System Architecture

We use the same system architecture as discussion in Section 4.1.1, on page 35. The New SLA proposed still considers the single service message instance.

5.2.2 A New IoT SLA

Consider an IoT cloud with M tenants; each tenant i, where $i = 1, \ldots, M$, sends X_i messages to the IoT broker service over a time T.

We propose to divide the total enforcement period (T) into K subperiods of equal time (T/K) as shown in Figure 5.1. We also propose that the provider needs to provide resources to process at least X_{ik} messages in a subperiod k for a tenant i. For simplicity, in our SLA, we consider $X_{ik} = X_i/K$. We state our new two-step SLA as follows:

> "*For a fee of \$Y to be paid by the tenant, the provider will supply an IoT message brokering service that will process a total of X_i IoT messages published or subscribed to an IoT message brokering service over a time T, and AT LEAST a total of X_i/K IoT messages over a time T/K.*"

Here our intuition is to do "input rate control". For example, the number of subperiods K can be made large enough to do input rate control at finer time scales. On the other hand, if $K = 1$, then

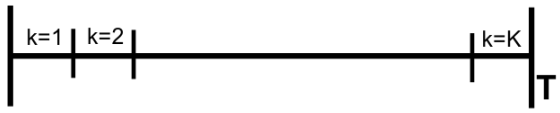

Figure 5.1 Splitting the total enforcement period into K subperiods.

the problem becomes the same as discussed in the previous chapter. We will show by simulation the effect of selecting different value of K in the evaluation section.

5.2.2.1 Analysis of SLA

The proposed two-step SLA retains the simplicity of statement and observability of the current SLA. It is still expressed in terms of volume of messages in a subperiod as well as in the total enforcement period.

Another major strength of proposed SLA comes from the introduction of the subperiod concept. The number of subperiods K offers an additional control for cloud providers to enforce proposed SLA. The cloud providers can distribute the total volume of messages over multiple subperiods and put an upper limit to messages in a subperiod. At the same time the number of subperiods offers cloud providers K chances to run a resource allocation algorithm. Another strength of the proposed SLA is that the tenant gets assurance on the number of processed messages in both periods T/K and T.

The major weakness of the SLA is the difficulty of its enforcement. Though the cloud provider can put the limit on message arrival in a subperiod, still in a subperiod the cloud provider has no knowledge of arrival of messages from the tenant. To counter this weakness, we have introduced the concept of additional enforcement period in our solution.

5.2.3 Definition of Key Terms

The majority of key terms used in formulation of our objective is already defined in Section 4.1.3, page 37. The additional definitions are given below.

Definition 1. **Number of subperiods** (K) gives the number of divisions of measurement period T.

Definition 2. **Additional enforcement period** (Δt): In this chapter we consider Δt as, additional enforcement period to process messages of previous *sub*period. Note that the definition of the variable is the same as in the previous chapter, but its use in the algorithm has changed.

5.2.4 Objectives

The objective remains the same as the one stated in the previous chapter.

5.3 Our Approach

5.3.1 Control Parameters and Constraints for Achieving SLA

As an additional control, we have introduced the concept of subperiods. The constraints remain the same as in the previous chapter.

5.3.2 Design Considerations

The design considerations for meeting proposed SLA are given below.

5.3.2.1 Subperiods

The subperiod parameter offers the following flexibility to the cloud provider in enforcing the proposed SLA:

(a) *The number of subperiods (K)* - a large value of K offers flexibility to distribute the message arrivals on many intervals, where as a small value reduces the complexity of running a resource algorithm. This tradeoff is discussed in the evaluation section of the chapter.

(b) *Resource guarantee on number of messages processed in k_{th} subperiod* - Providers can set the resource limit on guaranteed number of processed messages in a subperiod to be either static or dynamic. It helps providers to tackle the limitation of no knowledge regarding arrival patterns. For simplicity, we propose that the provider needs to provide resources to process at least X_i/K messages in each subperiod from a tenant i.

In this chapter, for simplicity, we have set the minimum resource requirement to number of processed message to be static and equal in all subperiods.

5.3.2.2 Additional Enforcement Time

As shown in Figure 5.2, if all the X_{ik} messages from a tenant i come arbitrarily close to the end of the k_{th} subperiod, conforming to SLA would require infinite resources. In order to handle this case, without the need to complicate the statement of the SLA with traffic arrival information, we propose to extend the enforcement subperiod to process messages of the previous measurement subperiod by an additional time, Δt.

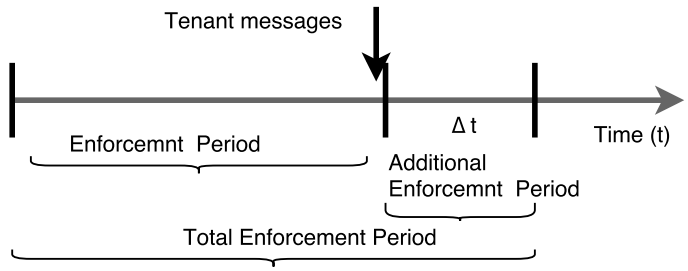

Figure 5.2 Measurement and enforcement time in a subperiod.

5.3.2.3 Service Over-provisioning

The design considerations about service over-provisioning are already discussed in the Section 4.2.2 on page 4.2.2.

5.3.3 Assumptions

System and traffic model assumptions are same as given in Section 4.2.3, on page 4.2.3.

5.3.4 Algorithm to Achieve the New SLA

Definition of variables. We denote the total number of tenants as M. We run the algorithm for K subperiods of duration T/K each. For each tenant i, the maximum number of allowed messages over each subperiod is M_i; the number of messages arrived and messages processed *up until* subperiod k is denoted by A_i and P_i respectively; the number of messages processed *during* subperiod k is denoted by P_{ik}; finally, the number of messages arrived *during* subperiod k is denoted by A_{ik}.

The variable D_i, defined in line 16 of the algorithm, denotes the minimum number of requests to be processed in the next (that is $(k+1)$-st) subperiod. It is computed at the end of subperiod k. More specifically, in the expression

$$D_i = \max\{0, \min[(X_i - P_i),\ (A_i - P_i)]\}, \tag{5.1}$$

the term $(A_i - P_i)$ represents how many messages from tenant i are still in the system. The term $(X_i - P_i)$ represents how many messages are still to be processed until the SLA is met. Note that this term may be negative, indicating that the system has processed more messages than the SLA guaranteed.

Control knobs. We propose to incorporate the following control parameters in the algorithm.

- *Over-provisioning capacity of the service instance.* The value of Δt decreases with increase in server capacity.

- *CPU scheduling.* As tenant arrivals are independent of each other, we propose per tenant CPU queueing. The set of all queues is denoted by Q. We propose a two-tier scheduling system, with (logically separate) "regular" and "overflow" portions in each tenant's queue. Regular queues are served in weighted round robin manner with the weight (w_i) for tenant i being equal to $X_i / \sum_{j=1}^{M} X_j$. If a tenant i has sent more that X_i / T messages during the enforcement period, the excess traffic is logically placed in the overflow queues. Regular queues get priority over overflow queues.

- *Policing.* We propose to exercise this control to limit the message input rate. The upper limit on a maximum number of messages can be a configurable parameter. This rate control is convenient to avoid overwhelming the server resources in case of high traffic.

Algorithm description. The mechanism to achieve the current SLA with the above mentioned extension is described in Algorithm 2. The proposed mechanism uses server over provisioning, policing, and WRR scheduling to serve the tenant queues.

5.3.4.1 Proof that the algorithm works.

We will show that when the algorithm completes its execution, $D_i = 0$ for all tenants.

From equation 5.1, $D_i = 0$ implies that

$$\min[(X_i - P_i),\ (A_i - P_i)]$$

is either strictly negative or 0. By definition, $(A_i - P_i)$ is nonnegative. If it is 0, all the messages that have arrived have also been processed. So, unless the tenant has sent less than X_i messages, the SLA is met; if it is strictly positive, the term $(X_i - P_i)$ is either strictly negative or 0. In both cases, the SLA is again met. On the other hand, $D_i > 0$ implies that $X_i > P_i$ and thus the SLA is not met.

Let's calculate next how the variable P_i increases. From line 15 of the algorithm,

$$P_i = \sum_k P_{ik} \tag{5.2}$$

where the sum extends over all K subperiods as well as the enforcement period. We will show that $P_{ik} > 0$ in any subperiod k in which there are messages in the queue of tenant i.

If, during any subperiod k, we have that $X_i \leq P_i$ (i.e., the SLA is already met) D_i is set to 0 in line 16. During the remaining subperiods and enforcement period, $X_i \leq P_i$ regardless of whether the

Algorithm 2 Over-provisioning of server capacity, WRR scheduling and rate limiting for tenant traffic

Output: Indication that the SLA was met or not, for each tenant.

```
Set D_i = 0, A_i = 0, P_i = 0, for all tenants i ∈ {1,...,M}.                [1]
for all subperiods k ∈ {1,...,K} do
    Set P_ik = 0, for all tenants i ∈ {1,...,M}.                             [3]
    Allow messages from a tenant i only if A_i < M_i then A_i = A_i + 1      [4]
    while CPU is free and ∃ a non empty queue in Q do
        if ∃ a set S ⊆ Q where P_ik < (X_i/K) + D_i then
            Run WRR scheduler for queues in S
            if tenant i is selected then P_ik = P_ik + 1
        else
            Run WRR scheduler for queues in Q
            if tenant i is selected then P_ik = P_ik + 1
        end if
    end while
    At the end of subperiod k
    ∀i ∈ {1,...,M}, set P_i = P_i + P_ik                                     [15]
    ∀i ∈ {1,...,M}, set D_i = max[0, min{(X_i − P_i), (A_i − P_i)]}          [16]
end for
At the end of the enforcement time,
for all tenant i ∈ {1,...,M} do
    if D_i = 0 then
        SLA met for tenant i                                                 [21]
    else
        SLA not met for tenant i
    end if
end for
```

WRR scheduler serves messages from this tenant or not. So, suppose that $X_i > P_i$ (i.e., the SLA is not yet met).

During the next subperiod, the WRR scheduler will serve this tenant until: (a) its queue is empty, or, (b) until $P_{ik} \geq (X_i/K) + D_i$. Therefore,

$$P_{ik} \geq \min(X_i/K, A_{ik}), \ \ \forall k \in \{1, \ldots, K\}. \tag{5.3}$$

From the last two equations, at the end of the enforcement period we have that $P_i \geq \min(X_i, \sum_k A_{ik})$. Regardless of which terms achieves the minimum, $D_i = 0$, q.e.d. △

5.4 Experimental Evaluation

5.4.1 Experimental Setup

The experimental setup considered in this evaluation is same as discussed in Section 4.3, on page 42.

5.4.2 Traffic Patterns

In the evaluation, we consider only worst case traffic pattern. We generate only worst case traffic patterns for a period where all X/K messages arrive in burst at the end of the subperiod. We consider this pattern as worst because provider has the least amount of time (compared to other traffic patterns) to process the message burst and meet the SLA. For this pattern, we assume the sensors burst all their measurements at the end of the subperiod in a burst of duration b. For this the instantaneous arrival rate, $AR(t)$, as, for all $t \in [0, T]$ is given below.

$$AR(t) = \begin{cases} (1/K) * X/b, & \text{for } \ (((i-1)T/(K)) - b) \leq t \leq iT/(K) \\ 0, & \text{otherwise.} \end{cases}$$

We select duration of burst as *burst size/bandwidth*, where bandwidth is fixed to 1 Mbps throughout the simulation.

5.4.3 Evaluation of Proposed Solution

We center the evaluation of our algorithm around three questions.

In order to visualize results only, since the algorithm is proven to satisfy the SLA, in the first question we **"verify that Algorithm 2 satisfies the SLA"**. For this, we run an experiment with one tenant, with values $X_1 = 1M$ messages, $T = 1$ month and overprovisioning factor $n = 2$. We selected

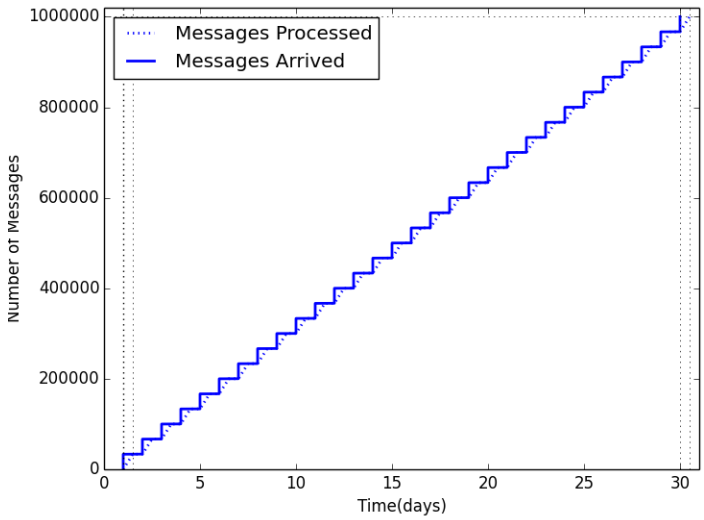

Figure 5.3 SLA Conformance.

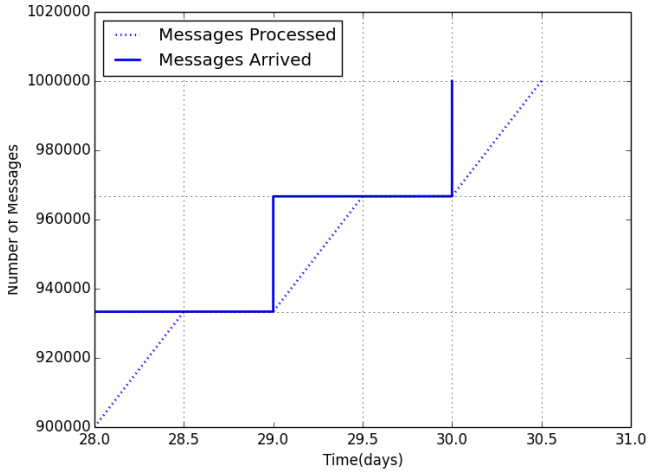

Figure 5.4 SLA Conformance.

the number of subperiods (K) to be 30 so that the subperiod interval is equal to one day. To explore the most severe case, we assume that the tenant sends all messages at the end of the subperiod. The number of messages in a sub period is set to 1/30 million. In Figure 5.4, we show that $X_1 = 1M$ messages are shown by the end of the additional enforcement period. In Figure 5.4, we have zoomed into the plot of Figure 5.3 to show that this period is equal to 12 hours, as expected.

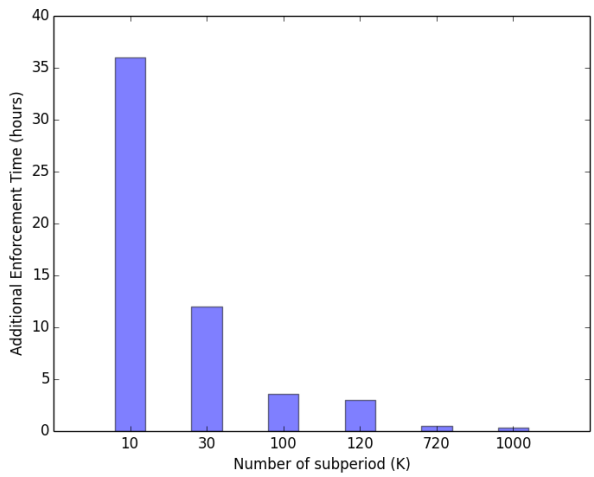

Figure 5.5 Effect of number of subperiods on additional enforcement time.

We then proceed to verify that **"by over-provisioning service capacity by a factor of n reduces the additional enforcement period to $T/(K\dot{n})$"**. For this, we run an experiment with one tenant, with values $X_1 = 1M$ messages, $T = 1$ month and overprovisioning factor $n = 2$; this gives us $T/n = 15$ days. We use the following six subperiod times: a) 1 day ($K = 30$), b) 6 hours ($K = 120$), c) 1 hour ($K = 720$) and $K\,10, 100, 1000$ in order to demonstrate the effect with an order of magnitude change in K. To explore the most severe case, we assume that the tenant sends all messages at the end of a subperiod. The number of messages in a sub period is set to $1/K$ million.

As shown in Figure 5.5, the additional enforcement period is inversely proportional to the number of subperiods.

The same effect is shown in Figure 5.6, in which we verify that **"by over-provisioning service capacity by a factor of n reduces the additional enforcement period to $T/(K\dot{n})$"**. We used $(K) = 30$, $X_1 = 1M$ messages, $T = 1$ month in this experiment and varied the overprovisioning factor from 2 to 100.

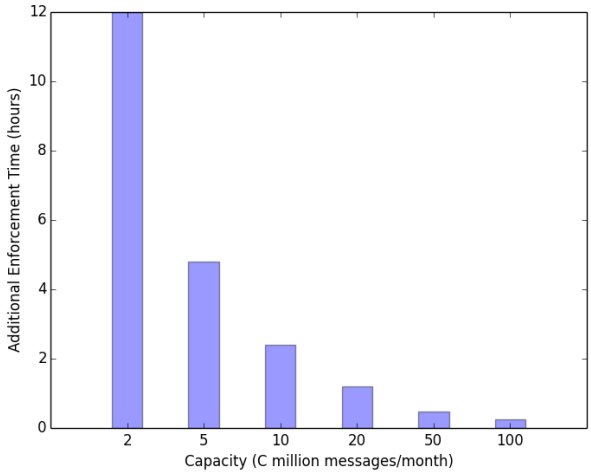

Figure 5.6 Effect of overprovisioning factor n on additional enforcement time.

5.4.4 Overhead and Tradeoff Analysis

Any increase in the number of subperiods K reduces the additional enforcement time, but at the cost of running the algorithm multiple times. We propose to select useful administrative value for example K=30 for T=1 month. A similar trade-off is to decide whether K should be dynamic or static. A dynamic K value would optimize the resource allocation but adds additional complexity to the algorithm.

In the case of large number of tenants, per-tenant queueing at the message broker server increases overhead. We can limit this overhead by adding more message broker servers and implementing a server load balancing mechanism. Since queues within the SLA limit get priority over queues with excess traffic, the variance in response time can be large. To limit variations in message response time, a dynamic weight scheduler can be implemented.

5.5 Conclusion

In this chapter, we proposed a novel SLA for IoT broker service. We studied conformance of the proposed, volume-based IoT SLA as a resource allocation problem. We proposed a buffering, scheduling and rate limiting mechanism to enforce such SLAs in a provable manner. Our evaluation shows that our solution achieves conformance within a fixed additional enforcement time period without the need to specify information about arrival patterns. We showed that the increase in number of subperiods and server capacity and decrease in the total measurement time reduces the length of

the additional enforcement period to conform to the SLA.

CHAPTER

6

RESOURCE ALLOCATION FOR CURRENT IOT SLA - CLUSTER OF SERVICE INSTANCES

This chapter is organized as follows. In Section 6.1, we discuss Server Scaling, the additional control knob and challenges associated with the presence of multiple servers. In the next section, we review some of the relevant literature on Server Scaling. In Section 6.3, we present the problem formulation. The problem is now substantially different, since three (conflicting) objectives are to be met. We discuss a plethora of controls in and define a solution that uses it in Section 6.4. In Section 6.5, we describe the experiments conducted and answer certain design questions. Finally, in Section 6.6, we conclude the chapter.

6.1 Service Clusters in Cloud Computing

6.1.1 Cluster of Service Instances in Cloud

In the cloud computing industry, a *service cluster* is defined as a logical grouping of *multiple* server instances.

The cloud provider creates this logical group once a tenants uses the cloud services for the first time. The motivation to deploy the cluster comes from a variety of reasons: a) tenant scale or performance. For example, when a single server cannot satisfy the tenant service requirement or performance need, and, b) business model. The provider uses multiple servers to offer a "pay as you go" service model to a tenant with elastic workload.

There are two major challenges for the provider in the case of cluster service deployment. First is the selection of a server in the cluster upon a new service request arrival. Second is the decision on how many servers in the cluster to keep running at any particular time instant.

The service cluster environment offers two important control knobs to the provider. The first control deals with "load balancing"; this control affects the performance of a tenant. We do not consider it in this thesis. We will discuss "server scaling", the second control in detail in the rest of this chapter.

6.1.2 Server Scaling in Cloud Computing

Server scaling refers to the addition (as well as reduction) of server resources in the cluster.

Server scaling can be implemented in two broad ways, "up" and "out". In the first, we increase the physical capacity of the server; for example, its Gflop rating or RAM capacity. Scaling up is a completely static approach; we do not consider it here. In the scale out approach, we "turn on/off" one or more server instances to increase/decrease capacity. This approach is more dynamic; for example, a physical server can be turned on in the order of minutes. A VM or a container can be turned on in an even smaller time interval. The decisions to turn a server on/off can be left to either the tenant or the provider. For example, a provider may shut down a few servers every night when no one is using them, and bring them up next morning. In this work, we focus on the scale out approach.

In a service cluster deployment, server scaling is an important control at the disposal of the provider, as it helps in striking the balance between tenant performance and operation cost. The provider faces following two main challenges in using the server scaling control: a) *when* to scale? and, b) *by how much* to scale?

Broadly speaking, the decisions to scale can be categorized as reactive, proactive and hybrid.

A reactive method reacts to system change, but does not anticipate it. For example, the decision trigger point can be based on exceeding a threshold for the current message queue size [16]. A proactive method anticipates the future needs and consequently adds resources or has them ready when they are needed [43], [58], [73]. A hybrid approach incorporates both proactive and reactive ideas. Generally speaking, the effect of a reactive scaling policy may not be immediate and may result into SLA violations. A predictive model may, on the other hand, be compute-intensive and waste resources. The broad tradeoffs between reactive and proactive approaches are done between these two metrics.

6.1.2.1 Server Scaling to Meet Current IoT SLA

We will investigate the use of scaling service instants out, in the rest of this chapter. The method is well suited for the environment of elastic load and the volume-based SLA we consider. More important, it is well suited for addressing the two additional objectives we introduce, besides the main objective of meeting the SLA. We discuss these objectives in Section 6.3.4.

6.2 Related Work in Resource Scaling in Cloud Computing

In this section, we survey related work in the area of server scaling. We group our survey in two broad scaling categories, *reactive* and *proactive* scaling. The definition of these categories was given in Section 6.1.2. The approaches and works differ generally in the following areas: a) KPI metric, b) SLA considered, c) workload characteristics, d) measurement, and, e) the control used.

6.2.1 Reactive Approaches

Commercial cloud providers offer reactive server scaling based on specific sets of rules [14], [74]. The policy scales resources based on a KPI metric, typically CPU utilization or average response time. Many researchers have considered rule- based scaling approaches to meet the SLA [65], [51], [45], [33].

In [65], the authors use CPU load, memory, bandwidth, storage and usage as KPI metrics. Scaling actions depend on the metrics exceeding prescribed thresholds. Actions include VM configuration change (scale up), VM migration, and VM addition (scale out). The authors do not include any particular SLA as their objective is to keep resources usage around 60%. The authors do not consider a burst traffic scenario. In [51], the authors also mention same objectives of keeping resource allocation around 70% and provide two thresholds, one to add and other one to reduce resources.

In [45], the authors used an idle CPU-relaetd metric and two thresholds to scale up and down, in order to meet a response time SLA. In addition, they recommend to use a "calm time" to let the scaling effect take action. They have considered two workload types: (a) transactional (e.g., e-commerce websites), and (b) batch (e.g., text mining, video transcoding, and graphical rendering). In [33], the authors consider response time SLA for web applications in a virtualized cloud computing environment. They measure the number of concurrent users, the number of active connections, the number of requests per second, and average response time per request on a VM; they scale the resources if these numbers cross given thresholds.

All of the above reactive works employ the same rule-based scaling policy as their solution. The above mentioned works have either response time SLA or an objective to keep resource utilization around a prescribed threshold. Our work in this chapter is different, as we consider a volume-based SLA along with average response time and cost to the provider as additional objectives.

6.2.2 Proactive Approaches

Prediction using various estimations techniques is very popular among researchers for resource management in elastic applications.

To model elastic applications and services, queueing theory has been extensively used in performance-related research. Queueing theory provides relevant metrics for scaling such as mean queue length, average wait time for requests under different arrival and service models. Urgoankar et al [83], used queueing theory in both the planning and operation phases (of ITIL) in order to estimate the resource requirements in a multi-tier environments; they investigated a response time SLA. In their approach, they used workload traces from a peak load scenario. Zhang et al [96] applied regression techniques for predicting the CPU demand, based on the number and type (browsing, ordering or shopping) of client requests and CPU load for each request; the objective was to reduce the wastage of resources and meet the SLA. Bacigalupo et al [19] used historical performance models and queueing theory to investigate resource management in enterprise cloud systems. Overall, the common element in these authors' queueing theory-based approach is the modeling of arrival patterns and the analysis of response time metric.

A control theory-based proactive approach [1], [2] requires a predictive algorithm in the controller to maintain the value of the controlled variable (e.g., CPU load) close to a desired level by adjusting the available control (e.g, number of VMs). Kalyvianaki et al [59] used Kalman filters to meet the response time SLA in a virtual server environment. The authors used prior data (several minutes worth) regarding the number of requests dropped in order to estimate the new amount of resources needed.

Machine learning is another promising method in dynamic resource allocation as it does not require prior knowledge. Machine learning-based approaches use training based on the observed load for better estimation of demand. The researchers [38], [20], [81] have taken this approach to scale VM resources to meet the response time SLA and provider cost objective. The major constraints of the approach are the training time required and learning failure in case of sudden burst type workload patterns.

In comparison to all of the above efforts, in our work the KPI is SLA is different (from the common response time SLA). Our proposed (dynamic) solutions fall under the reactive regime.

6.3 Problem Formulation

6.3.1 System Architecture

The topology considered in this chapter is given in Figure 6.1. In our system, we consider a cluster of message servers to process tenants request coming from multiple gateways. The detailed description of system elements in this topology is already given in section 3.1, page 29.

6.3.2 IoT SLA

In this chapter, we focus on the same volume-based IoT SLA discussed in the earlier chapters of the thesis. We restate it here:

> "*For a fee of $Y to be paid by the tenant, the provider will supply an IoT message brokering service that will process a total of X IoT publish or subscribe messages over a time period T.*"

Processing a message may include tasks such as authentication, logging in a database, analytics, etc. The tasks may be running on separate physical servers and thus the switch fabric may be crossed several times during the processing of a single message. The exact details of processing are out of the scope of this thesis. We abstract these details by assuming a (constant) time t_{proc} secs for each message.

6.3.3 Definition of Key Terms

Some of these definitions we have already seen in Chapter 4, Section 4.1.3; we repeat them here for convenience.

Definition 1. **Measurement Period** (T): this is the time interval, specified in the SLA statement, that is used to measure the number of message arrivals. △

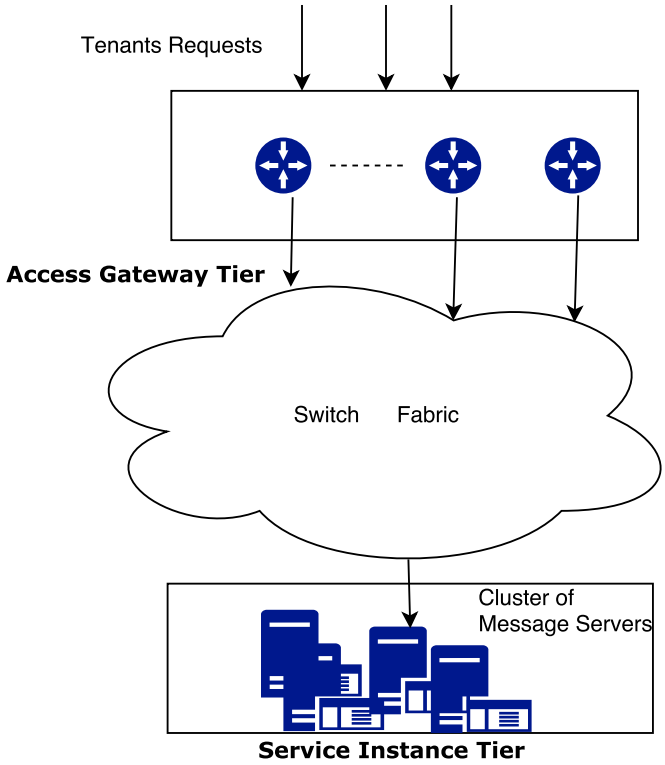

Figure 6.1 Topology considered: Cluster of service instances (message servers) for all tenants.

Note that this value is universally applied to all tenants. In evaluations presented in previous chapters, we used a typical value of one month for T. We present in Section 6.5.3.1 reasoning for also using $T = 1$ day.

Definition 2. **Message processing capacity** (N): this is the number of servers the provider must set aside for a tenant. △

We discuss factors that the provider can consider in selecting a suitable value for N in Section 6.5.4.1, page 89.

Definition 3. **Enforcement Period** (EP): this is the time interval needed to *unconditionally* guarantee that the SLA is met. △

We have:

$$EP = T + \Delta t \tag{6.1}$$

The provider then can guarantee that X messages will be processed by time EP.

There is, of course, a direct relationship between N and Δt; we assume that the provider decides on what is a reasonable value for Δt and determines N, the number of servers allocated to the tenant. How are these servers brought into action is the subject of the scaling policy we define next. We discuss the relationship in more detail in Section 6.5.4.1, page 89.

Definition 4. **Scaling Policy**. Formally, a scaling policy p is identified by the N-tuple

$$\{t_1(p), t_2(p), \ldots, t_N(p)\}, \tag{6.2}$$

where $t_i(p) = t \in [0, T]$, if the policy activates server i during the measurement period at time instant t and $t_i(p) = \infty$ if the policy never activates server i during the measurement period. △

Loosely speaking, a scaling policy (or simply policy) p is any rule that specifies the time instant at which the N servers will be activated.

Note that, implicit in the definition is the assumption that, once activated, a server will remain activated till the end of the SLA enforcement period. If the policy allowed *scaling down* (that is deactivation of servers) as well, an N-tuple as described in Equation 6.2 would not be sufficient to formally define the policy. Mixed scaling up and scaling down policies are left for future study.

Definition 5. **Tenant performance index** $CPI(p)$: even though the SLA does not mention any metrics regarding performance, average message delay is still of interest to the tenant. If two scaling policies both meet the SLA and one gives the tenant messages half the (average) delay, clearly one is more preferable than the other.

Let $n_T(p)$ denote the number of messages processed by time EP, under policy p.

Let A_j denote the time instant the j-th message arrived in the system. Let $F_j(p)$ denote the time instant the system finished processing the j-th message.

Then

$$D_j(p) = F_j(p) - A_j \tag{6.3}$$

denotes the delay incurred by the j—th message.

The quantity

$$CPI(p) = \frac{1}{n_T(p)} \sum_{j=1}^{n_T(p)} D_j(p) \tag{6.4}$$

is the average delay incurred over the entire measurement (plus enforcement) period, when policy p is in effect. For completeness, we define $CPI(p) = 0$ when $n_T(p) = 0$.

Definition 6. **Provider cost** $PC(p)$: the operational cost for a cloud provider includes a variety of factors like power, cooling, IT support, and maintenance.

We focus here on a simpler definition of cost that takes into account the amount of time a server is on. This time depends, of course, on the policy p. Let $F(p)$ denote the time instant at which all X messages are processed. We define $pc(p,i)$, the cost of activating server i under policy p as:

$$pc(p,i) = \max[F(p) - t_i(p), 0] \tag{6.5}$$

Recall that, in Equation 6.5, we assume that, once server i is activated at time $t_i(p)$, it will remain on until the end of the maximum possible time to process all tenant messages (that is, $F(p)$). We make this assumption for simplicity - this way, we avoid having to define policies to scale down, i.e., turn the servers off.

With the above definition of server cost, $PC(p)$, the provider cost incurred by policy p is then given by Equation 6.6:

$$PC(p) = \sum_{i=1}^{N} pc(p,i) \tag{6.6}$$

6.3.4 Objectives

We have to deal with three objectives. The main one (O1) is to meet the SLA defined in Section 6.3.2. Note that this is the only objective stemming from the provider-tenant contract. The second and third objectives (O2) and (O3) recognize that a policy incurs a cost to the provider that cannot be high and at the same time provides delay to the tenant that should be "good".

Design a scaling policy p in order to:
"(O1) Meet the SLA"
"(O2) at a reasonable cost to the provider (as given by Equation 6.6)"
"(O3) while giving reasonable delay performance to the tenant (as given by Equation 6.4)"

Intuitively, a given policy p cannot be "good" for both the tenant as well as the provider. Reducing the $CPI(p)$ metric in Equation 6.4 requires activating a server i earlier (i.e., decreasing $t_i(p)$); this will inadvertently increase the cost $PC(p)$ in Equation 6.6. A tradeoff must be made between the two conflicting objectives. It is for this reason that we do not state objectives (O2) and (O3) as minimizations of cost and delay metrics. The main reason for stating the objectives vaguely (reasonable is not defined quantitatively) is due to the fact that the SLA does not specify any traffic characteristics.

In Section 6.5.5.1, we discuss the tradeoff issues in detail, after we have defined policies explicitly.

6.4 Our Approach: Design of Scaling Policies to Achieve the SLA

6.4.1 Introduction and Motivation

Clearly, a policy will affect all three objectives. We denote the set of all policies as $\mathscr{P}$. Such policies must meet objective (O1). One can design policies to "favor" the tenant objective (O3) or the provider objective (O2).

Obviously, there is an infinite number of policy designs - static or dynamic. Our goal is to *systematically* select subsets of policies for evaluation. These subsets should contain policies that can be "fine-tuned" for tradeoffs between objectives (O2) and (O3). We explain this in more detail with the examples in Sections 6.4.3 and 6.4.4.

We state our assumptions next.

6.4.2 Assumptions

In the following, we make the (reasonable) assumption that the tenant will send the full amount of traffic specified in the SLA. The case of sending less traffic is not very interesting; the case of sending traffic in excess of the contractual amount X is more challenging and cannot be covered by the SLA statements as well as the policies we need to design to meet this SLA. We leave this case for future study.

We also make a "service affinity" assumption: any message can be processed by any server. For a broker service, this is a realistic assumption to make. For other IoT services (for example analytics), the assumption may not hold true, if the server must keep "state" that is to be preserved from message to message.

6.4.3 Specific Designs: Static Policies

We denote the set of all possible static policies as $\mathscr{P}_{st}$.

The rationale for considering such policies is the zero measurement overhead they incur; another reason for considering their design is the low configuration overhead, as we discuss in more detail in Section 6.5.5.1.

Loosely speaking, a static policy p chooses the server activation instants according to some rule that does not change throughout the operation of the system (e.g., it does not depend on the traffic pattern). We give a few examples next.

6.4.3.1 A static policy that favors the tenant.

The static policy p_1, defined in Equation 6.7, pre-allocates all N servers at time 0.

$$t_i(p_1) = 0, \quad i = 1, \ldots, N. \tag{6.7}$$

In the notation of Equation 6.2, we have

$$p_1 = \{0, 0, \ldots, 0\} \tag{6.8}$$

Rationale. Intuitively, such a policy "favors the tenant"; it provides the best possible delay performance, since all servers are available to process messages the moment they arrive. Objective (O3) is met in the best possible way; that is, no other policy (in the entire set $\mathscr{P}$ of all possible policies, not just static) can achieve a lower value for the $CPI(p)$ metric in Equation 6.4. On the other hand, the provider cost attains its maximum value; that is, no other policy (in the entire set $\mathscr{P}$ of all possible policies, not just static) can incur a higher value for the cost $PC(p)$ in Equation 6.6. This should be apparent from Equations 6.5 and 6.7.

Policy p_1 will trivially meet objective (O1), since all N servers are available since the beginning. In the worst case, this will happen by the end of the Enforcement Period defined in Equation 6.1.

6.4.3.2 A static policy that favors the provider.

The static policy p_2, defined in Equation 6.9, allocates all N servers at time T.

$$t_i(p_2) = T, \quad i = 1, \ldots, N. \tag{6.9}$$

In the notation of Equation 6.2, we have

$$p_2 = \{T, T, \ldots, T\} \tag{6.10}$$

Rationale. Intuitively, such a policy "favors the provider". Objective (O2) is met in the best possible way; that is, no other policy (in the set $\mathscr{P}_{st}$ of all possible static policies) can achieve a lower value for the cost $PC(p)$ in Equation 6.6. This should be apparent from Equations 6.5 and 6.7. On the other hand, the tenant cost attains its maximum value. This can be seen from Equation 6.3, in which the finishing time $F_j(p)$ for each message j is maximum under $p = p_2$.

Note that improvements on the provider cost are possible, at least in theory, if one considers *dynamic* policies that take into account the number of messages that actually arrived in $[0, T]$; see Section 6.4.4.6.

Policy p_2 will trivially meet objective (O1), since all N servers are available at the end of the measurement period. This will happen by the end of the Enforcement Period defined in Equation 6.1.

6.4.3.3 A policy that stages scaling.

The static policy p_3, defined in Equation 6.11, will activate the servers in a "staged" fashion, every T/N units of time.

$$t_i(p_3) = (i-1)T/N, \quad i = 1, \ldots, N. \tag{6.11}$$

In the notation of Equation 6.2, we have

$$p_1 = \{0, T/N, 2T/N, \ldots, (N-1)T/N\} \tag{6.12}$$

Rationale. Intuitively, this policy does not favor either the tenant or the provider as extremely as policies p_1 or p_2 do. It is, therefore, a tradeoff between the tenant and provider objectives. Note, however, that this policy does not provide a knob to "fine-tune" this tradeoff.

Policy p_3 will trivially meet objective (O1), since all N servers are available at the end of the measurement period. This will happen *at most* by the end of the Enforcement Period defined in Equation 6.1.

6.4.3.4 Periodic policies

This is a subset of the set $\mathscr{P}_{st}$ of static policies. We denote it as $\mathscr{P}_{st-per}$.

A periodic policy with period parameter K. A periodic policy p_4, with period parameter K is defined via:

$$t_1(p_4) = 0, \quad t_{i+1}(p_4) = t_i(p_4) + T/K, i = 1, \ldots, N-1. \tag{6.13}$$

In the notation of Equation 6.2, we have (assuming, to be exact, that $t_N \leq T$)

$$p_4 = \{0, T/K, 2T/K, \ldots, (N-1)T/K\} \tag{6.14}$$

The parameter K can be any positive real number. For simplicity of evaluation, we will consider integer values only. Note that, based on the value of K, a periodic policy may activate all servers before T or it may leave some servers idle by time T. The former case occurs when a value $K \geq N$ is chosen; the latter occurs when $K < N$.

Rationale. This set of policies provides a knob to the designer to strike a tradeoff between the tenant and provider objectives. When K is large, p_4 favors objective (O3); it becomes policy p_1 for $K = \infty$. On the other hand, when $K = N$, it coincides with the staging policy p_3 in Equation 6.11.

Note that for values of K smaller than N, the policy will not activate all N servers by time T. Thus it will only guarantee meeting objective (O1) for $K \geq N$.

6.4.3.5 A random policy.

The static policy p_5, defined in Equation 6.15,

$$t_i(p_5) = t(i), \quad i = 1, \ldots, N, \tag{6.15}$$

where $t(i)$ is a number in $[0, T]$ will activate the servers in a "random" fashion. The values of $t(i)$ can be selected at the beginning of the measurement period. This policy will activate all N servers by the end of the measurement period, and thus it will meet the SLA. Intuitively, p_5 provides a knob to strike a tradeoff between the tenant and provider objectives. Since the values of $t(i)$ are known, the designer can tell which objective will be favored.

6.4.4 Specific Designs: Dynamic Policies

6.4.4.1 Introduction and Motivation

The rationale for considering such policies is the potential they give to achieve the tenant/provider objectives for arbitrary traffic patterns. The obvious drawback is the measurement and processing (as well as configuration) overhead they incur; we detail this in Section 6.5.5.

We denote the set of all dynamic policies as $\mathscr{P}_{dyn}$. Note that there is "no structure" in this set - dynamic policies can be defined in numerous ways. In the sequel, we will evaluate the behavior of dynamic policies only in certain subsets that possess clearly defined structure. Loosely speaking, one such subset contains policies that aim to improve tenant performance.

The pattern of server activation, will, in general, be more complicated than the case of static policies; for example, it may depend on the traffic pattern, the state of the system, etc. Guaranteeing that a dynamic policy will meet objective (O1) should be examined carefully. Towards this end, it suffices to make sure in the design of a dynamic policy that by time T it will have activated enough servers to process however many messages are left in the system by time EP.

6.4.4.2 Policies that favor the tenant objective.

Let d_{th} denote a positive real number; the idea is to try and keep the average delay of a tenant (see Equation 6.4) below this threshold. The designer can then use d_{th} as a knob to directly address objective (O2). We will discuss how to choose d_{th}, based on X, T and N in Section 6.5.4.3.

Let $n_t(p)$ denote the number of messages processed by time t, $t \in [0, T+\Delta t]$, when policy p is in effect. Let $D_j(p)$ denote the delay incurred by the j-th message, as described in Equation 6.3.

The quantity

$$Del(t) = \frac{1}{n_t(p)} \sum_{j=1}^{n_t(p)} D_j(p), \tag{6.16}$$

is the calculated, current average delay the (processed) messages experience up to time t. For completeness, we define $Del(t) = 0$ when $n_t(p) = 0$.

Note that there may be messages waiting or in process that have not departed by time t. For simplicity, these messages are not taken into account in calculating this index.

An entire class of tenant performance improving policies can be defined using the triplet

$$\left(d_{th}, \{t_k^d\}, R\right),$$

where

1. d_{th} is the above-mentioned threshold;
2. R is the rule for activating servers; and,
3. $\{t_k^d\}$ denote the decision instants in $[0, T]$ at which R is applied.

R uses the calculated index $Del(t_k^d)$ in Equation 6.16 and the threshold d_{th}. Note that the number of decision instants is arbitrary, but there can be up to N such instants at which a server will be activated. At the k-th decision instant t_k^d, a (potentially empty) set of servers is activated (and kept on till the end of the current enforcement period).

We present three specific designs next.

6.4.4.3 The set of periodic improvement policies $\mathscr{P}_{slow}$.

The decision instants t_k^d are chosen in an arbitrary, periodic fashion, giving rise to a class of policies. The rule $R1$ for activating servers is given by Algorithm 3.

This set provides two knobs to the designer: the threshold d_{th} and the choice of decision instants t_k^d. An example policy in the set $\mathscr{P}_{slow}$ is p_6, defined by

Algorithm 3 Rule $R1$

Activate one server at time 0.
if $Del(t_k^d) > d_{th}$ **then**
  activate one more server, if possible
else
  do not activate any server.
end if

$$\tilde{t}_k^d = kT/N, \quad k = 0, 1, \ldots, N-1. \tag{6.17}$$

Rationale. The rationale for this policy is that no server (except the first) will be activated until (current) tenant performance deteriorates below the acceptable threshold. Then the system attempts to improve performance by allocating the smallest possible amount of resource - 1 server.
Will objective (O1) be guaranteed? As defined so far, p_6 may fail to meet objective (O1).

Consider, for example, the following scenario. Set $d_{th} = T/N$, with the goal of giving the tenant good average delay. Suppose that messages start arriving after the last decision instant $\tilde{t}_{N-1}^d = (N-1)T/N$. The delay calculated at all decision instants is zero, so no server will be activated in this scenario. We therefore augment the definition of the policy with a "safeguard" activation at time $\tilde{t}_N^d = T$:

Algorithm 4 Rule $R1.1$

Rule $R1$ applies.
At time $\tilde{t}_N^d = T$: Activate all servers.

With this activation, enough servers are available to finish processing a total of X messages by the end of the enforcement period, guaranteeing objective (O1).

With the augmentation defined in Algorithm 4, we can formally define a typical, dynamic policy in $\mathscr{P}_{slow}$ as:

$$p_6 = (d_{th}, \{\tilde{t}_k^d\}_{k=0}^N, R1.1) \tag{6.18}$$

6.4.4.4 The set of periodic improvement policies $\mathscr{P}_{fast}$.

The decision instants t_k^d are chosen in an arbitrary, periodic fashion, giving rise to a class of policies. The rule for activating servers is given by Algorithm 5.

Algorithm 5 Rule $R2$

```
Activate one server at time 0.
if Del(t_k^d) > d_th then
    activate two more servers, if possible
else if Del(t_k^d) > d_th then
    activate one more server, if possible
else
    do not activate any server.
end if
```

This set provides two knobs to the designer: the threshold d_{th} and the choice of decision instants t_k^d. An example policy in the set $\mathscr{P}_{fast}$ is p_7, defined by

$$t_k^d = kT/N, \quad k = 0, 1, \ldots, N-1. \tag{6.19}$$

Rationale. The rationale for this policy is that no server will be activated until (current) tenant performance deteriorates below the acceptable threshold. Then the system attempts to improve performance by allocating server resources more aggressively than a policy in the set $\mathscr{P}_{slow}$, based on how "bad" the current performance is.

Will objective (O1) be guaranteed? As defined so far, p_7 may fail to meet objective (O1) for the same reason we discussed in Section 6.4.4.3; the same scenario applies.

Algorithm 6 Rule $R2.1$

```
Rule R2 applies.
At time t̃_N^d = T: Activate all servers.
```

With this activation, enough servers are available to finish processing a total of X messages by the end of the enforcement period, guaranteeing objective (O1).

With the augmentation defined in Algorithm 6, we can formally define a typical, dynamic policy

in $\mathscr{P}_{fast}$ as:

$$p_7 = (d_{th}, \{\tilde{t}_k^d\}_{k=1}^N, R2.1) \tag{6.20}$$

6.4.4.5 Nonperiodic improvement policy p_8.

The decision instants t_k^d are chosen to be *arrival instants*. The rule for activating servers is given by Algorithm 7.

Algorithm 7 Rule $R3$

```
Activate one server on the first arrival instant.
if Del(t_k^d) > d_th then
    activate one more server, if possible
else
    do not activate any server.
end if
```

This policy provides only one knob to the designer: the threshold d_{th}.

Rationale. The rationale for this policy is that no server will be activated until (current) tenant performance deteriorates below the acceptable threshold. Then the system attempts to improve performance by reacting to bursts of arrivals. A burst of N arrivals during a period in which current performance is bad will activate all servers.

Will objective (O1) be guaranteed? Yes, with the augmentation specified in Algorithm 8.

Algorithm 8 Rule $R3.1$

```
Rule R3 applies.
At time t̃_N^d = T: Activate all servers.
```

Let $t(k)$, $k = 1, \ldots, N$, denote the first N message arrival instants; let $t(N+1) = T$. Finally, we can formally define policy p_8 as:

$$p_8 = (d_{th}, \{t(k)\}_{k=1}^{N+1}, R3.1) \tag{6.21}$$

6.4.4.6 Can dynamic policies improve the provider objective?

The static policy p_2 in Equation 6.9 has the lowest provider cost among all static policies. Can this cost be further reduced, however, *by a dynamic policy*?

The hope is that such a policy will not need to activate all N servers. Activating server i before time T will increase the corresponding individual cost $pc(p,i)$ in Equation 6.5, but the total cost in Equation 6.6 *may* have less than N terms.

So, let's (arbitrarily) consider a dynamic policy that observes t_X, the time instant in $[0,T]$ the X-th message arrives. The policy decides the number of servers to be activated based on the observed, actual traffic pattern. Suppose that $L \leq N$ servers are to be activated at time t_X. Formally, the dynamic policy p_9 is defined as follows:

$$t_i(p_9) = \begin{cases} t_X, & i = 1,\ldots,L, \\ \infty, & i = L+1,\ldots,N. \end{cases} \tag{6.22}$$

All L servers will be busy until the time instant t_L at which an amount of messages m'_L (strictly less than L and possibly 0) is left for processing. Let

$$m_L = \left\lfloor \frac{X}{L} \right\rfloor.$$

Then

$$t_L = t_X + m_L \cdot t_{proc}$$

The remaining m'_L messages will be processed in time t_{proc}; thus, from Equation 6.5, we have

$$F(p_9) = m_L \cdot t_{proc} + \mathbf{1}(m'_L \neq 0) \cdot t_{proc},$$

where $\mathbf{1}(A)$ is equal to 1 if condition A is true and 0 otehrwise. By a similar argument, if we define

$$m_N = \left\lfloor \frac{X}{N} \right\rfloor,$$

we have

$$F(p_2) = m_N \cdot t_{proc} + \mathbf{1}(m'_N \neq 0) \cdot t_{proc}.$$

Finally, the provider costs under the two policies are

$$PC(p_9) = L \cdot m_L \cdot t_{proc} + L \cdot \mathbf{1}(m'_L \neq 0) \cdot t_{proc}, \quad PC(p_2) = N \cdot m_N \cdot t_{proc} + N \cdot \mathbf{1}(m'_N \neq 0) \cdot t_{proc}.$$

One can easily construct numerical examples in which

$$PC(p_9) < PC(p_2)$$

(One such example, albeit with unrealistic numbers, is: $N = 7, L = 5$ and $X = 100$.) On the other hand, one can easily construct numerical examples in which

$$PC(p_9) > PC(p_2).$$

(Take $N = 10, L = 7$ and $X = 100$.)

The subtlety in the above calculations comes from the fact that, in Equation 6.6, *all servers* are considered on and contribute to the cost calculations until the last message is served. In realistic SLAs, with X in the order of millions, the cost differential will be negligible.

In summary, we say that *no dynamic policy can significantly outperform the static policy* p_2, in terms of provider cost. It is possible, however, to improve the tenant performance objective (O2), since for some traffic patterns, policy p_9 may activate all N servers at a time t_X strictly less than T. In this case, processing will finish at a time instant strictly less than $T + \Delta t$. When evaluating policies, we make the implicit assumption that policy p_9 activates all N servers at a time t_X.

6.4.5 Summary: the Selected Sets of Policies

In the next section, we propose to evaluate the following sets of policies:
(a) First, the two policies p_2 and p_1 will provide a "baseline" for comparison of all policies, since they optimize (at least among static policies) objectives (O2) and (O3) respectively.
(b) The set of static, periodic policies $\mathscr{P}_{st-per}$. The period parameter K is a knob for tradeoffs between the two objectives.
(c) The set $\mathscr{P}_{rule}$ that contains the dynamic policies $\{p_6, p_7, p_8\}$. These policies provide several knobs for the tradeoffs, as we explain in more detail in the next section.
(d) Policy p_9.

6.5 Experimental Evaluation

We center the simulation-based evaluation of our proposed scaling policies around the three objectives (O1 - meet the SLA), (O2 - reasonable cost to the provider) and (O3 - while giving reasonable delay performance to the tenant) mentioned in Section 6.3.4, page 66.

We organize our experiments into two broad groups. In the first one, presented in Section 6.5.3,

we investigate how a specific policy fares with respect to the three objectives. In the second group, presented in Section 6.5.3, we answer specific design questions and thus compare policies to one another.

Before we present the results, we describe our experimental setup and the traffic patterns we have selected for the evaluation.

6.5.1 Experimental Setup

The experimental setup considered in this evaluation is similar to the one used in Section 4.3, page 42. The main difference is the presence of multiple servers as indicated in Figure 6.1, page 64. In all experiments, we use $t_{proc} = 0.001$ sec.

6.5.2 Traffic Patterns

In the evaluations, we consider three traffic categories, in order to capture a variety of IoT sensor monitoring scenarios: continuous, periodic and event-based measurements.

Continuous measurements are produced by sensors which do not have local storage and thus send their measurements to the datacenter immediately. Periodic measurements are produced by sensors or access concentrators which do have local storage and thus send their measurements periodically. Finally, event-based measurements are produced by sensors which are activated when a trigger (e.g., an accident) occurs. We generate traffic patterns in each category by defining the instantaneous arrival rate, $AR(t)$, as indicated below, for all $t \in [0, T]$.

1. **Continuous Measurements.** We assume that the sensors send their measurements uniformly over the time interval $[0, T]$. Therefore,

$$AR(t) = X/T, \quad \text{for } 0 \leq t \leq T.$$

2. **Periodic Measurements.** The sensors group measurements for K hours and then burst them in a burst duration b. Therefore, for the i-th such burst,

$$AR(t) = \begin{cases} (1/K) * X/b, & \text{for } (((i-1)T/(K)) - b) \leq t \leq iT/(K) \\ 0, & \text{otherwise.} \end{cases}$$

An hourly burst is a special case of periodic measurement where T= 24 hours, and K = 24.Therefore, for the i-th hour burst,

$$AR(t) = \begin{cases} X/b, & \text{for } (((i-1)24/K) - b) \leq t \leq i24/K \\ 0, & \text{otherwise.} \end{cases}$$

In our evaluation, we have considered $K = 10, 24$, and 100.

3. **Event-based Measurements.**

   The sensors burst their measurements after a trigger event occurs, in a single burst that is sent at time t_1. Therefore,

$$AR(t) = \begin{cases} X/b, & \text{for } (T - t_1) \leq t \leq (T - t_1 + b) \\ 0, & \text{otherwise.} \end{cases}$$

   We have considered two such traffic patterns in our evaluations.

   (a) $t_1 = 0$, for a burst at the start.

   (b) $t_1 = T - b$, for a burst at the end.

6.5.3 Evaluation of Policies Within a Given Class Only

In this section, we evaluate how a given policy meets the three design objectives.

6.5.3.1 Evaluation of the base static policies p_1 and p_2

We start by establishing a "baseline"; we evaluate the two static policies p_1 (best for the tenant objective) and p_2 (best for the provider objective) given in Equation 6.7, page 68, and Equation 6.9, page 68. These policies can be used to compare the performance of all others.

We have used the following simulation parameters in evaluating policies p_1 and p_2.

- Measurement time (T) is set to 1 day. This choice is applicable to any IoT system for which operation (including taking measurements, transporting and analyzing them) makes more sense on a daily basis.

- We set the number of servers (N) to 10. This choice gives us enough service instances to see (substantial) differences in the actions of the designed policies. (With only two servers available, for example, a dynamic policy with tens of decision instants may allocate both of them "early" and essentially stay inactive most of the time.) We justify this choice in Section 6.5.4.1.

- We set the burst size as X/B, where B is the number of bursts within the measurement period. We select duration of burst as *burst size/bandwidth*, where bandwidth is fixed to 100 Mbps throughout the simulation.

Objective (O1). To meet the SLA, the provider should be able to process all X messages arrived in T (in 1 day) by time $T+\Delta t$. We verify with Figure 6.2 for policy p_1, and Figure 6.3 for policy p_2, that the SLA is met for all traffic patterns. In this experiment, we have set $X = 10M$ messages and thus $\Delta t = X \cdot t_{proc}/10 = 0.278 \approx 0.3$ hours; we expect that, in the worst case, processing should be finished at time 24.2778.

As shown in Figure 6.2, the time for processing X messages under policy p_1 depends on the traffic pattern. Going from top left down the left column and then from top right down the right column, the exact finishing times are 24.278, 24.011, 0.278, 24.027, 24.003, and 23.99 hours respectively. They are all less than the the theoretical 24.2778.

The time for processing X messages is independent of the pattern, as shown in Figure 6.3, for policy p_2; it is equal to 24.278 hours.

Objective (O2). Next we investigate the effect of the two policies on objective (O2) - the cost to the provider (as given by Equation 6.6, page 66).

In Figure 6.4, we graph the cost incurred under different traffic patterns for both policies p_1 and p_2. In the case of policy p_2, the cost is the same for all traffic patterns; this is because we process messages at time T, irrespective of arrivals. Whereas, in case of policy p_1, messages finish processing at different time instants, based on the traffic pattern; hence we get different provider cost, as would be expected from Equation 6.5. In the case of early burst, the cost is same for both policies p_1 and p_2.

Objective (O3). Next, we investigate the effect of policy on objective (O3) - the tenant's average delay performance (as given by Equation 6.4, page 65).

In Figure 6.5, we graph the average delay experienced under different traffic patterns with policies p_1 and p_2. As expected, policy p_1 gives (much, much) better performance than policy p_2 for all traffic patterns. In case of policy p_1, the delay performance varies from 0.001 seconds (for uniform traffic) to approximately 500 seconds (burst at the end), whereas for policy p_2 the delay performance varies from approximately 500 seconds (burst at the end) to approximately 90,000 seconds (burst at the start).

6.5.3.2 Evaluation of the periodic policies in $\mathscr{P}_{st-per}$

In this section, we evaluate p_4, a static periodic policy in the set $\mathscr{P}_{st-per}$ given in Equation 6.14, page 69. With the values of K chosen in this experiment, the policy is guaranteed to meet the SLA; we focus our evaluation on Objective (O2) and Objective (O3) only.

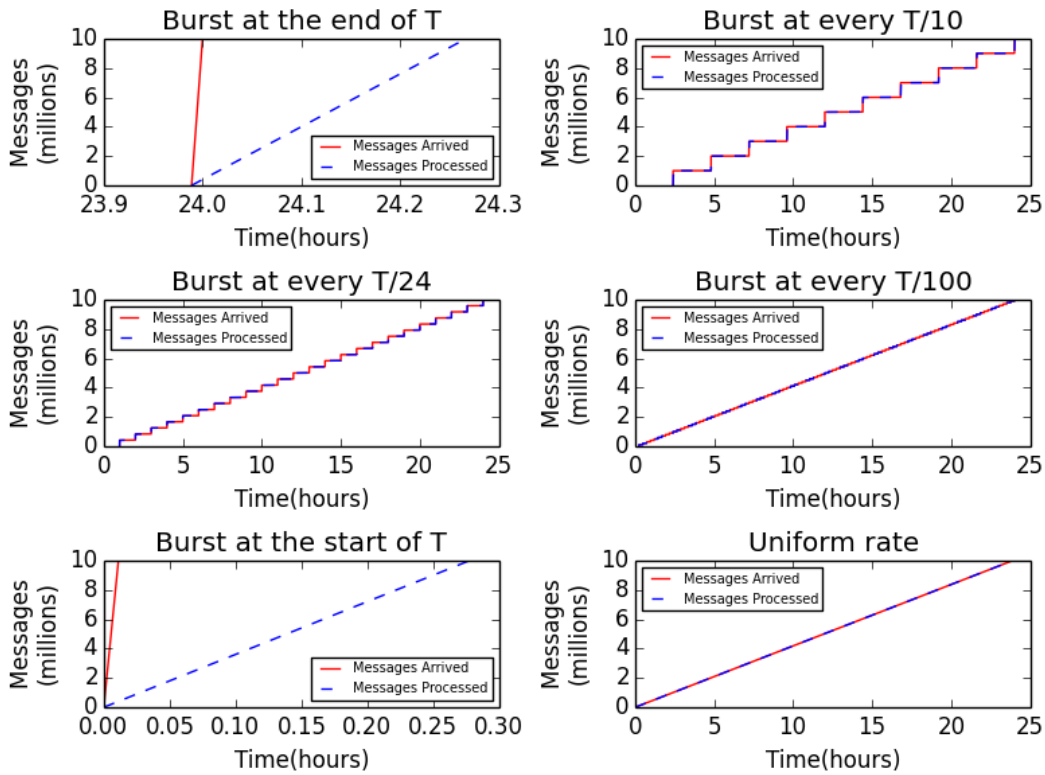

Figure 6.2 Objective (O1) for different traffic patterns with policy p_1, favoring tenant.

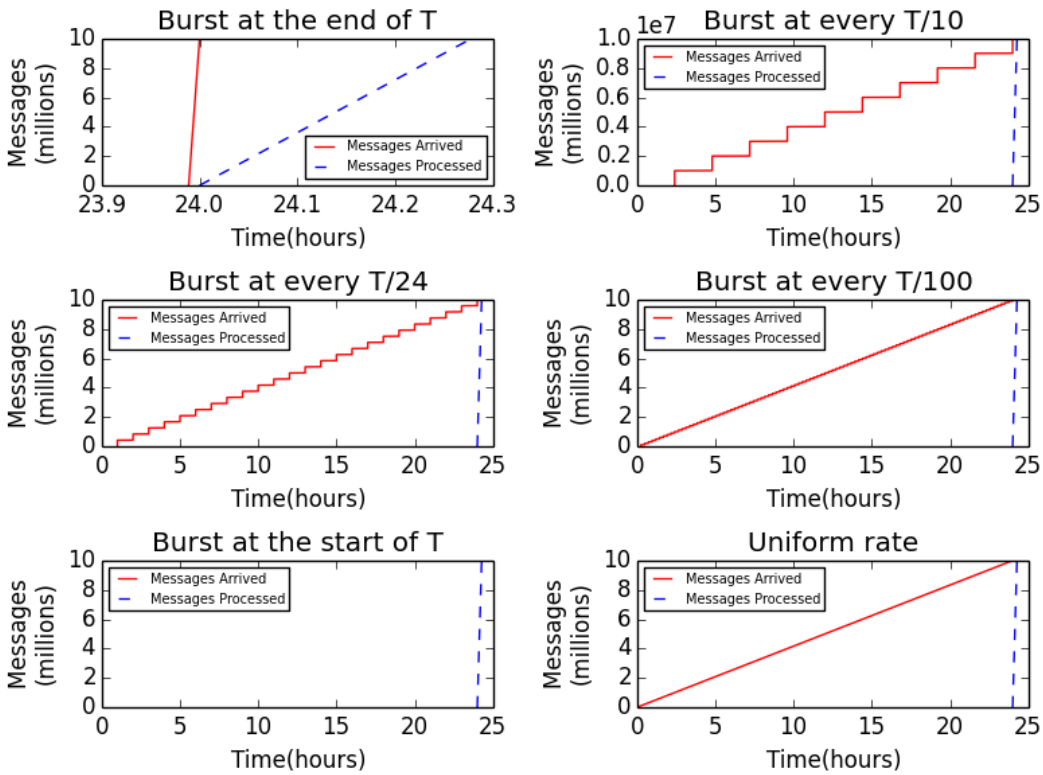

Figure 6.3 Objective (O1) for different traffic patterns with policy p_2, favoring provider.

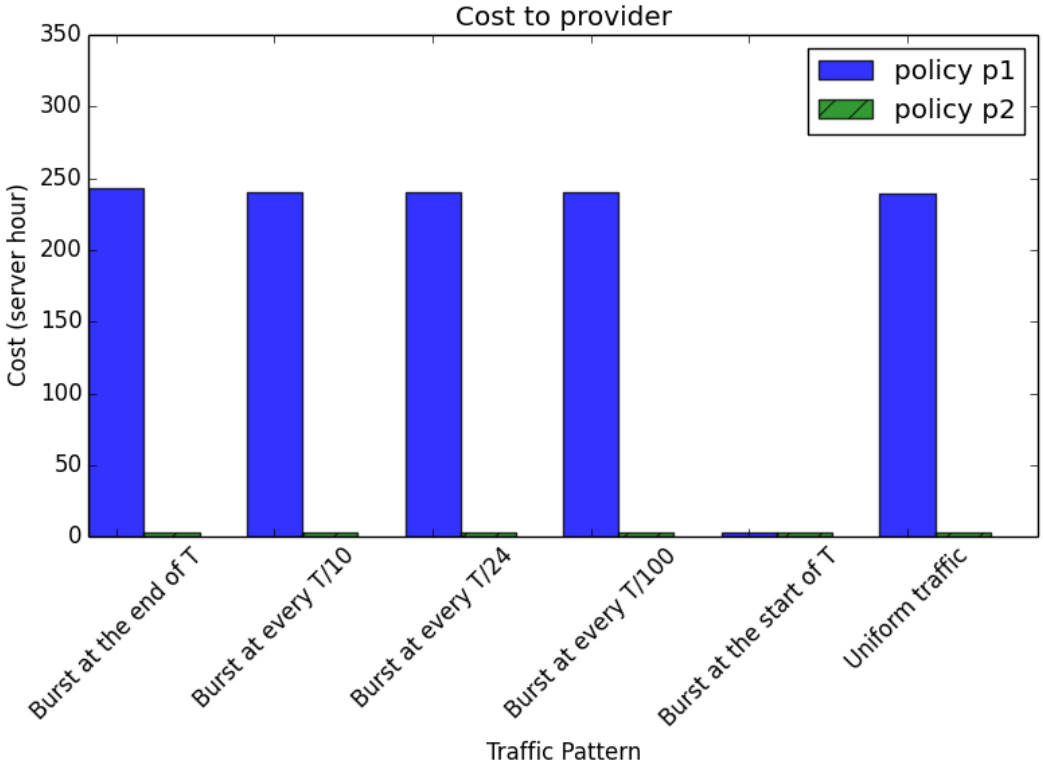

Figure 6.4 Objective (O2) for different traffic patterns with policies p_1 and p_2.

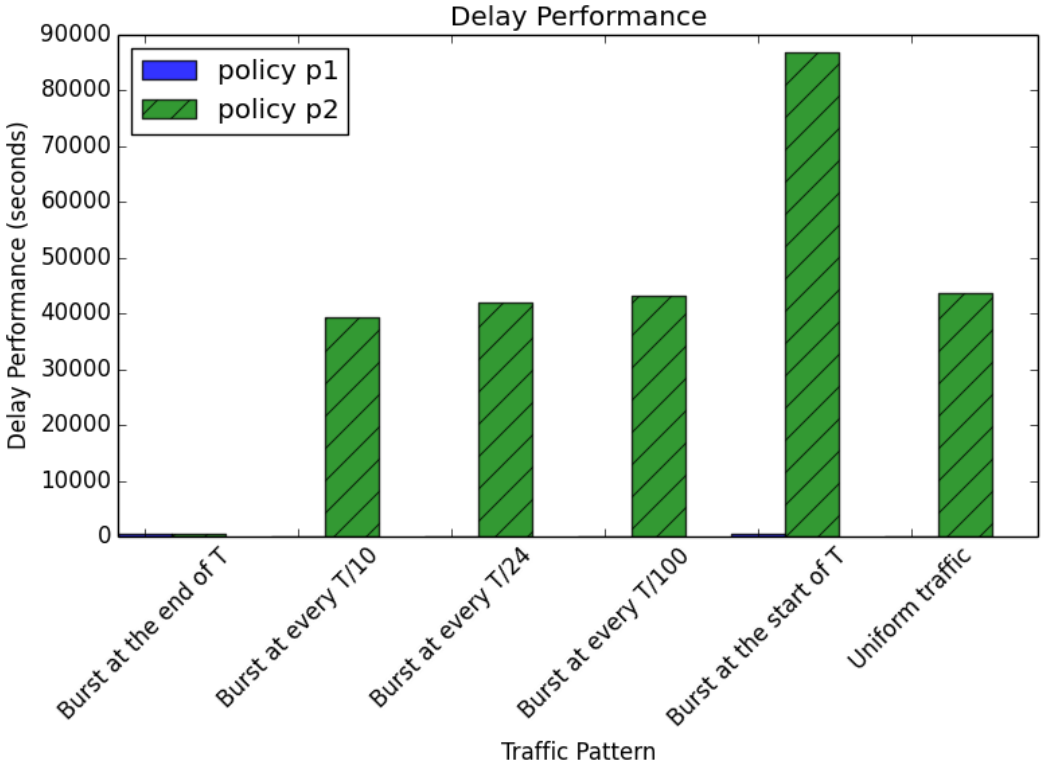

Figure 6.5 Objective (O3) for different traffic patterns with policies p_1 and p_2.

For the evaluation of this policy, we have considered three values of the parameter K - 10, 20 and 100. These values have been selected for the following reasons: first, as defined, a policy in $\mathscr{P}_{st-per}$ will only guarantee meeting the SLA for $K \geq N = 10$. Second, for a large value of K, the policy becomes similar to policy p_1.

The remaining simulation parameters are the same as in Section 6.5.3.1.

Objective (O2). First we investigate the effect on objective (O2) - the cost to the provider (as given by Equation 6.6, page 66).

In Figure 6.6, we graph the cost to provider incurred under different traffic patterns for different values of K. As expected, smaller values favour the provider. From the graph we can observe that for a particular value of K, the provider cost is approximately same for all traffic patterns (except for the traffic pattern 'Burst at the start of T', as all the messages are processed way before $T = 24$ hours (at 2.59, 1.99, and 1.03 hours for K=10, 20, 100 respectively).

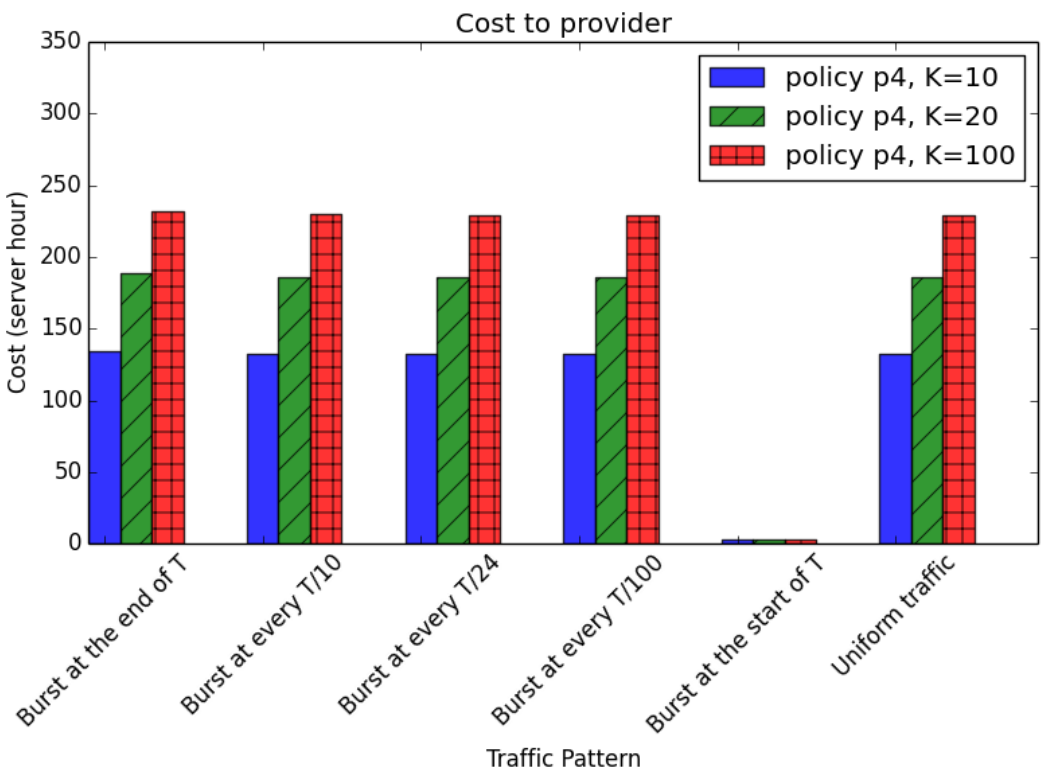

Figure 6.6 Objective (O2) for different traffic patterns for static periodic policy p_4 with periods $K = 10, 20, 100$.

Objective (O3). Next, we investigate the effect on objective (O3) - the tenant's average delay performance (as given by Equation 6.4, page 65).

In Figure 6.7, we graph the average delay experienced under different traffic patterns for policy p_4 with different values of K. As expected, higher values give better performance for traffic patterns distributed (multiple burst or uniform) over a day period. For $K = 20$, all 10 servers are available at time $t = 1.2$ hours (20 periods in 24 hours) and the average delay varies from approximately 1 millisecond (continuous traffic) to just over a minute (for burst every $T/10$) for traffic patterns distributed over a day period. If we increase K to 100, all 10 servers are available after 0.24 hours and the average delay varies from approximately 1 millisecond (continuous traffic) to 48 seconds (for burst every $T/10$) for traffic pattern distributed over a day period. The average delay performance with $K = 100$ is very close to best case average delay performance given by policy p_1.

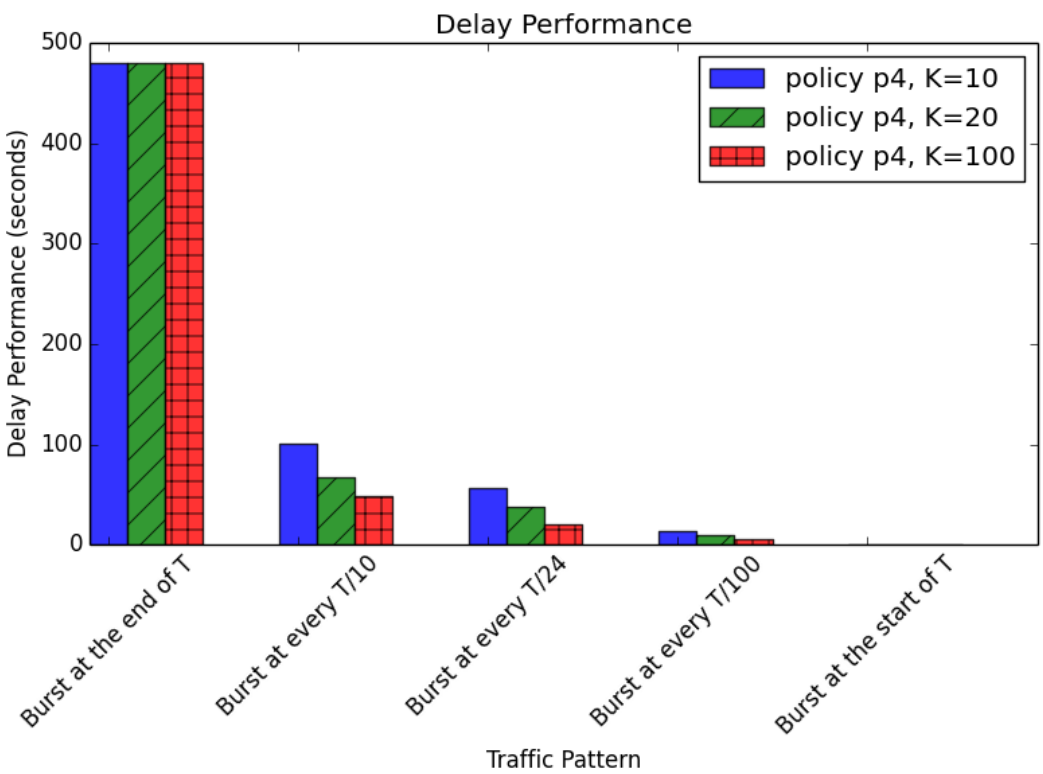

Figure 6.7 Objective (O3) for different traffic patterns for static periodic policy p_4 with periods $K = 10, 20, 100$.

Overall, from Figures 6.7 and 6.6, we observe that the parameter K can be used to strike a tradeoff between the tenant and provider objectives. When K is large, policy p_4 favors objective (O3); and when K is small, it favors objective (O2). With $K = 100, N = 10$, the 10-th server will be activated at about 2.4 hours.

6.5.3.3 Evaluation of the dynamic policies in $\mathscr{P}_{rule}$

In this section, we evaluate dynamic policies in the set $\mathscr{P}_{rule}$ defined in Section 6.4.4, page 70. We focus our evaluation on Objectives (O2) and (O3). We also evaluate the effect of d_{th} on the two objectives.

The simulation parameters are the same as in Section 6.5.3.1.

Objective (O2). First, we investigate the effect of policies on objective (O2) - the cost to the provider (as given by Equation 6.6, page 66).

In Figure 6.8, we graph the cost to provider incurred under different traffic patterns for policies p_6, p_7, and p_8, with the delay threshold d_{th} set to 60. As shown in the figure, for the traffic pattern labeled 'Uniform traffic', all policies cost the same to the provider as *no scaling is performed at all*. Recall that the service time of a message is 0.001 seconds and uniform arrival interval is 0.00864 seconds. A queue never forms and the $Del(t_k^d)$ never exceeds the threshold. The reason for no scaling is that a single server suffices to serve the load. We exclude this traffic pattern from evaluation discussion.

Policy p_7 favours tenant more than policy p_6 whenever the threshold is reached for fast increase as defined in rule $R1.1$ on page 72. As a tradeoff, provider incurs a larger cost with policy p_7 as compared to policy p_6, whenever fast increase is possible. For the traffic patterns labeled 'burst at every T/10' and 'burst at every T/24', the burst duration is high enough to cross the d_{th} threshold and policyp_7 activates the fast increase rule. Both policies incur the same cost, if conditions for fast increase are not met or the traffic arrives after the $(N-1)*T/N$ time instant. This happens in the remaining traffic patterns.

Policies p_6 and p_7 activate the first server at time $t = 0$; policy p_8 activates the first server only upon the first arrival. This explains the higher cost for periodic policies for the traffic pattern labeled 'Burst at the end of T' as shown in Figure 6.8. For all other traffic patterns, policy p_8 incurs a larger provider cost; it reacts to bursts and activates more than one server at a decision instant trying to keep average performance below the threshold.

Objective (O3). Next, we investigate the effect on objective (O3) - the tenant's average delay performance (as given by Equation 6.4, page 65).

In Figure 6.9, we graph the average delay experienced under different traffic patterns for dynamic

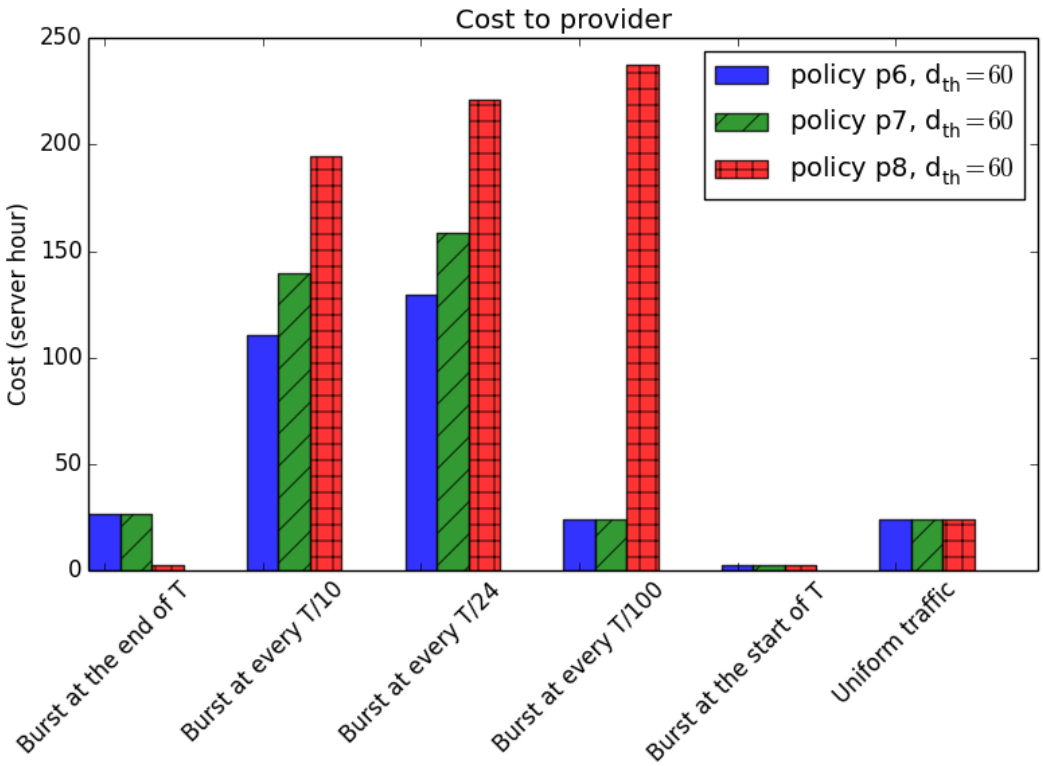

Figure 6.8 Objective (O2) for different traffic patterns with dynamic policies p_6, p_7, and p_8 in $\mathscr{P}_{rule}$ with delay threshold $d_{th} = 60$.

periodic policies p_6, p_7, p_8 with the delay threshold d_{th} set as 60.

For traffic patterns labeled 'burst at every T/10' and 'burst at every T/24', policy p_7 activates more servers than p_6 and hence provides obvious better performance. For the pattern labeled 'burst at the end of T', both policies activate the safeguard mechanism. For other traffic patterns, the d_{th} is never reached.

The dynamic policy p_8 gives better average delay performance in comparison to p_6, p_7, as it reacts on arrival for all the traffic patterns. Our burst size is large enough to activate all the servers if needed, hence achieves the obvious improvement over other policies.

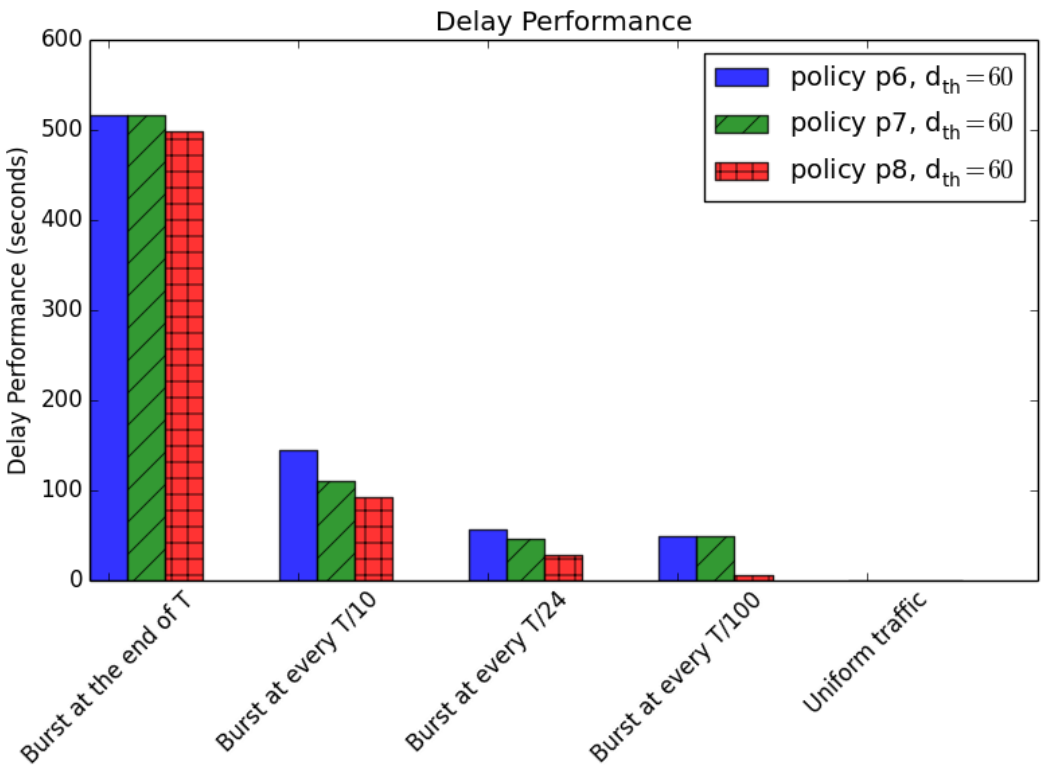

Figure 6.9 Objective (O3) for different traffic patterns with dynamic policies p_6, p_7, and p_8 in $\mathscr{P}_{rule}$ with delay threshold $d_{th} = 60$.

Overall, the dynamic policy set defined is tilted in favour of tenants. Any aggressive rule in favour of tenant like fast increase in policy p_7 and activation on arrival in policy p_8 improves the performance of tenants but incurs a very high cost for the provider.

6.5.4 Answering Specific Design Questions

6.5.4.1 How can one select the value of the parameter N?

As we mentioned in Definition 2, Section 6.3.3, page 63, N is the number of servers the provider must set aside for a tenant.

One criterion the provider can use to select N, is the desired length of the additional enforcement period Δt. Using this criterion implies that the provider places a *strong emphasis on the guarantee* and designs for the worst case traffic scenario.

Suppose, for clarity of argument, that $X = 10M$ messages; consider three values for processing times, $t_{proc} = 10^{-1}, 10^{2}, 10^{3}$ μsec. The total processing load on one server is approximately $PL_1 = 1$ second, $PL_2 = 16$ minutes and $PL_3 = 3$ hours respectively. Regardless of the value of the measurement period, T, one server alone could finish processing of such a burst in the calculated example times. When $T = 1$ hour, PL_1 may be deemed acceptable by most tenants, in which case even $N = 1$ would suffice. PL_3 would likely irritate even the most understanding tenants.

Therefore, given the total load that the tenant brings to the system, a suitable value for N can be chosen from the relationship[1]

$$\Delta t = \frac{X \cdot t_{proc}}{N} \tag{6.23}$$

Note the similarity to the overprovisioning factor n in the single server case (see, for example, Equation 4.2, page 39).

In most of the experiments of Section 6.5.3, we arbitrarily chose the value $N = 10$. Let's consider one additional scenario, on an extreme side of the spectrum of choices.

We take $N = 100$ as a representative of a high overprovisioning cost to the provider. For this choice, the dynamic scaling policies provide enough room to play with. We have used dynamic policy policy $\mathscr{P}_{fast}$ in our evaluation, as it can activate N servers way before time T, where as policy $\mathscr{P}_{slow}$ can activate one server every period.

In Figure 6.10, we plot the tenant objective for various traffic patterns, for policy p_7. The corresponding provider cost is shown in Figure 6.11. In order to see differences in tenant performance more clearly, we remove one traffic pattern in Figure 6.12. In the patterns with multiple bursts, the large value of N gives more options to activate additional servers; hence the better performance we see (at the expense of more cost to provider, of course). For such patterns, the range of delay performance is 3 to 55 seconds for $N = 100$ and 37 to 348 seconds with $N = 2$ seconds. The cost to provider

[1] Under the light of our discussions in Section 6.4.4.6, Equation 6.23 is not entirely accurate, as it ignores the fact that, when less that N messages remain for processing, not all servers will be busy. As we argued in that section, we can ignore this slight inaccuracy for all practical purposes.

is better with $N = 2$ (approximately 36 server hours) as compared to $N = 100$ (approximately 515 to 1700 server hours.)

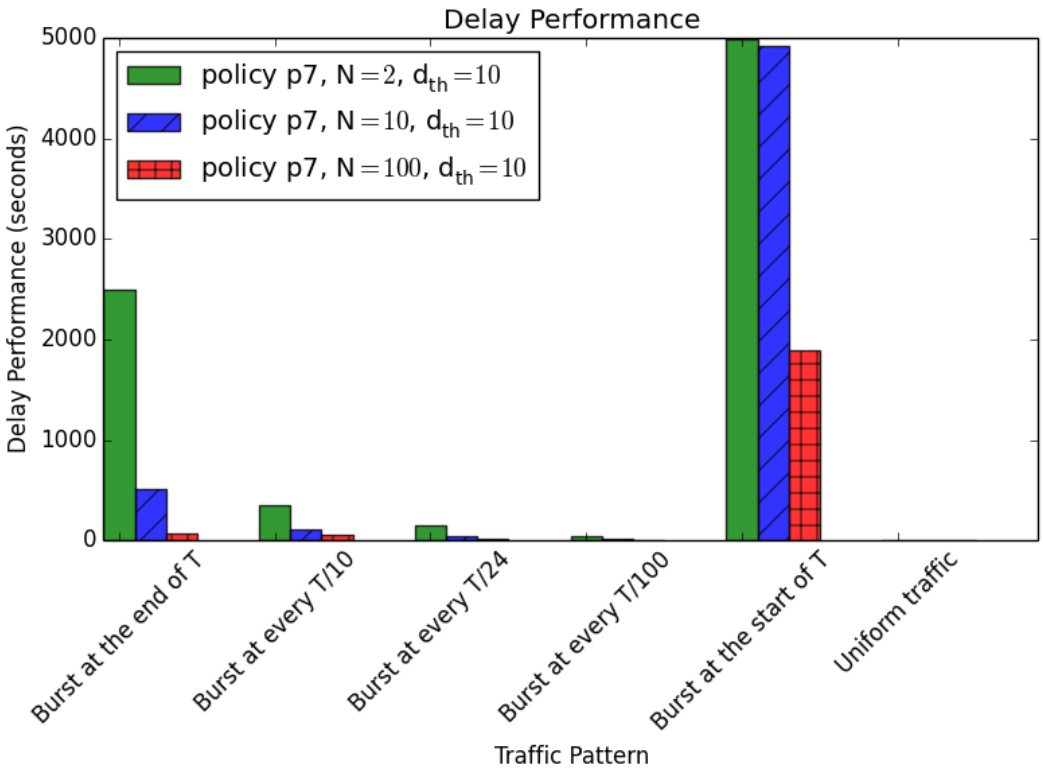

Figure 6.10 Effect of N on Objective (O3) for different traffic patterns, policy p_7.

6.5.4.2 What policy is the "best" for a given traffic type?

We have answered this question in pieces in our evaluation in Section 6.5.3. We simply summarize the findings here.

For the provider objective (O2), as we have argued in Section 6.4.3.2, page 68, the static policy p_2, defined in Equation 6.9, that allocates all N servers at time T is the best among all possible static policies. For the same objective, as we have argued in Section 6.4.4.6, page 75, the dynamic policy p_9 in Equation 6.22, that allocates all N servers at the time t_X of the last arrival is slightly better, slightly worse or as good as p_2. Since it provides better performance with respect to objective (O3),

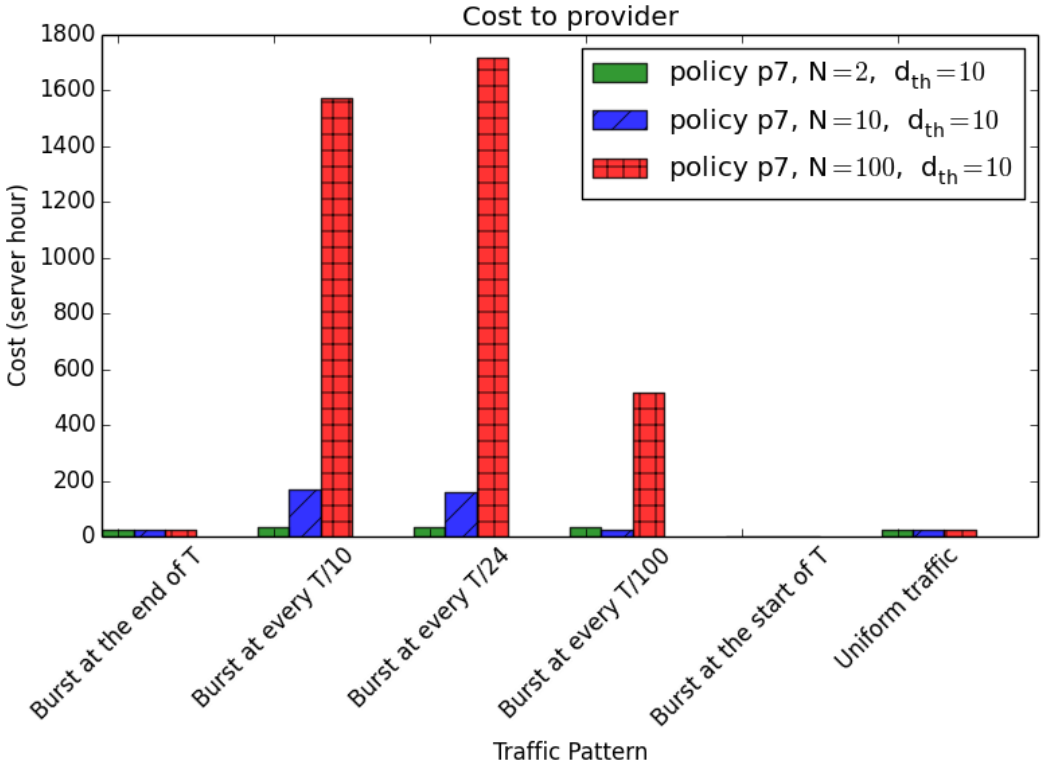

Figure 6.11 Effect of N on Objective (O2) for different traffic patterns, policy p_7.

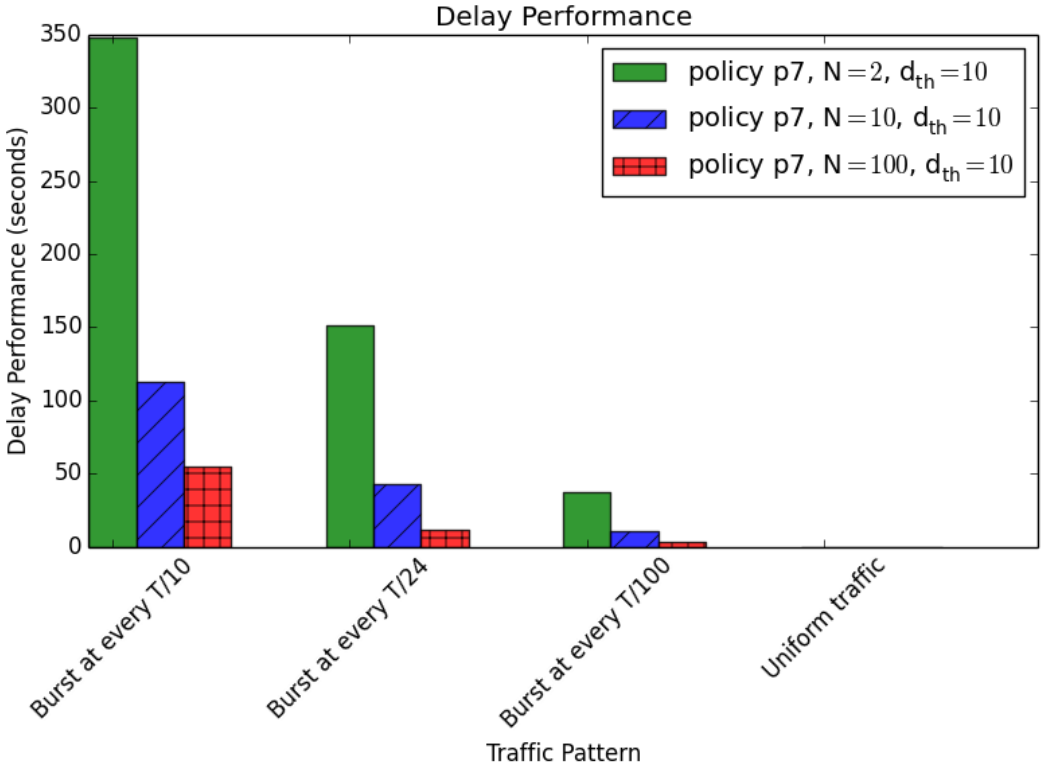

Figure 6.12 Effect of N on Objective (O3) for different traffic patterns except single burst, policy p_7.

we consider it the "best" policy for the provider objective (O2).

For the tenant objective (O3), as we have argued in Section 6.4.3.1, page 68, the static policy p_1, defined in Equation 6.7, that pre-allocates all N servers at time 0 is the best among all possible policies, static or dynamic. A simple argument, along the lines of comparing policies p_2 and p_9 for the provider objective (O2), shows that a dynamic policy that activates all servers at the time of the first arrival, instead of time 0, is as good as policy p_1, with a lower cost for the provider. We consider it the "best" policy for the tenant.

6.5.4.3 How can one choose d_{th}?

The selection of values for d_{th} is always a challenging task for the provider as delay changes with different traffic patterns. Overall, a large threshold value is against the objective of favouring tenant whereas a small value of threshold can result into scaling all N servers quite fast.

In this section, we select the d_{th} values as 10, 60 and 100. One justification for these choices comes from the evaluation of static periodic policies where the average delay for most of the traffic patterns is below 100 units.

In our simulation, the patterns with large burst sizes (single burst, or burst at every $T/10$) create a very high average delay just after the burst, so the selected d_{th} values do not make an impact. Therefore, we focus our evaluation of d_{th} selection with traffic patterns 'burst at every $T/24$' and 'burst at every $T/100$' only. The selected traffic patterns have comparatively smaller burst sizes, so the impact of d_{th} selection is visible under some policies as shown in Figures 6.13, 6.14, 6.15 and 6.16.

In Figures 6.13 and 6.14, we graph the cost to provider incurred with different dynamic policies for the two selected traffic patterns. In both figures, the policy p_8 does not react to the selected d_{th} values. Policy p_8 does not have a "cool off period for next activation", so it ends up activating all N servers. In both figures, policy p_6 incurs the least cost for even smaller value of d_{th}. As shown in Figure 6.13 for $d_{th} = 10$, policy p_7 reacts more aggressively than policy p_6, causing a larger cost to the provider.

In Figures 6.15 and 6.16, we graph the delay performance. The impact of d_{th} is visible in Figure 6.16 for the value $d_{th} = 10$, due to the more aggressive rule in server activation.

In conclusion, the d_{th} impact will be limited with large burst sizes. Dynamic policies should include a "cushion period" for next activation to adjust to the burst. Second, an inappropriately chosen d_{th} value renders a "dynamic" scaling policy a "no scaling" policy. d_{th} is a dangerous knob!

In another set of experiments, we examined the effect of d_{th} within a policy. For this, we chose policies in $\mathscr{P}_{fast}$ and varied d_{th} with $N = 10$ and $N = 100$. In Figure 6.18 ($N = 10$), and

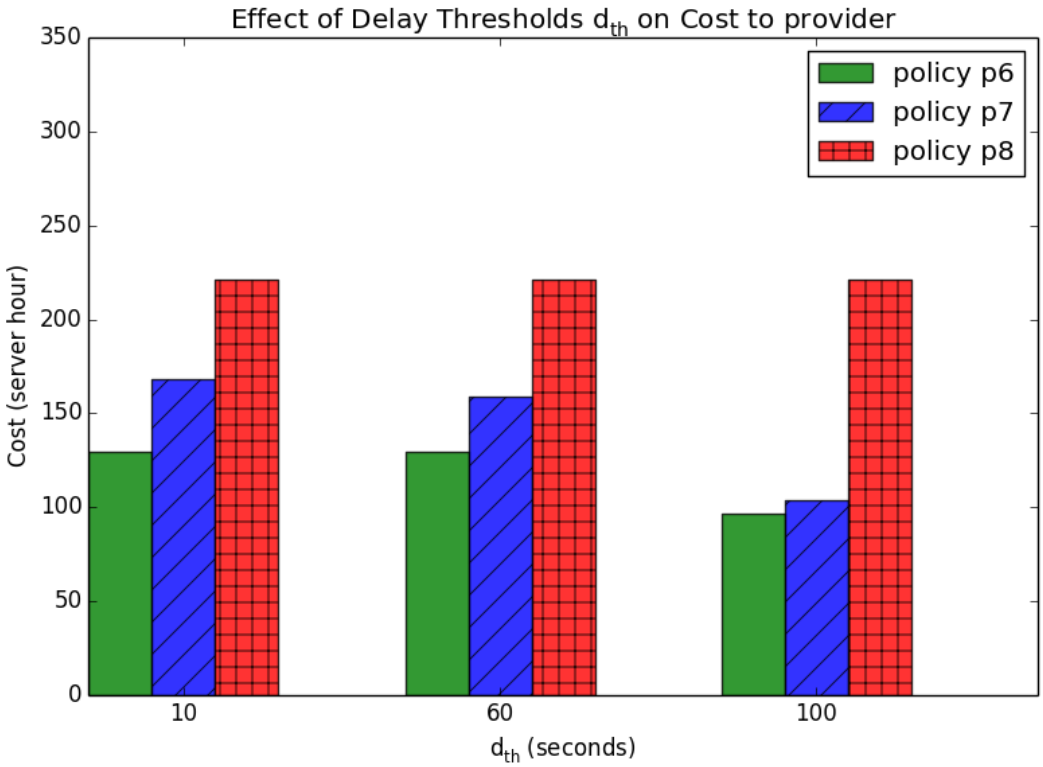

Figure 6.13 Effect of d_{th} on Objective (O2) for traffic pattern 'burst at every $T/24$'.

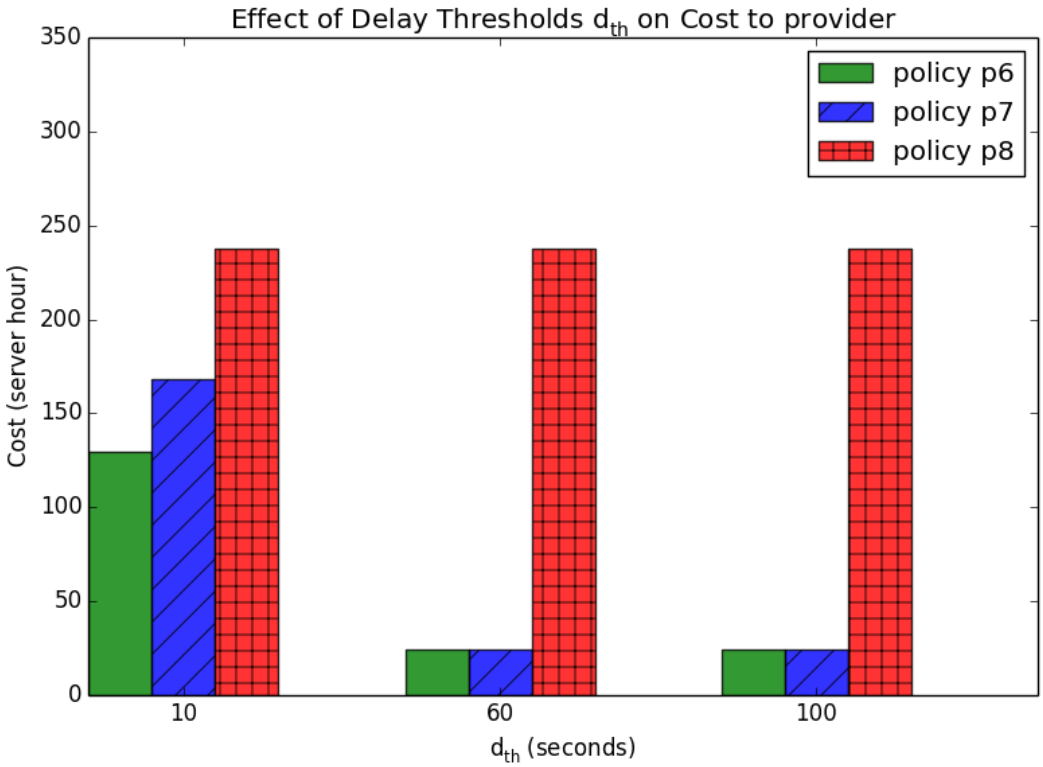

Figure 6.14 Effect of d_{th} on Objective (O2) for traffic pattern 'burst at every $T/100$'.

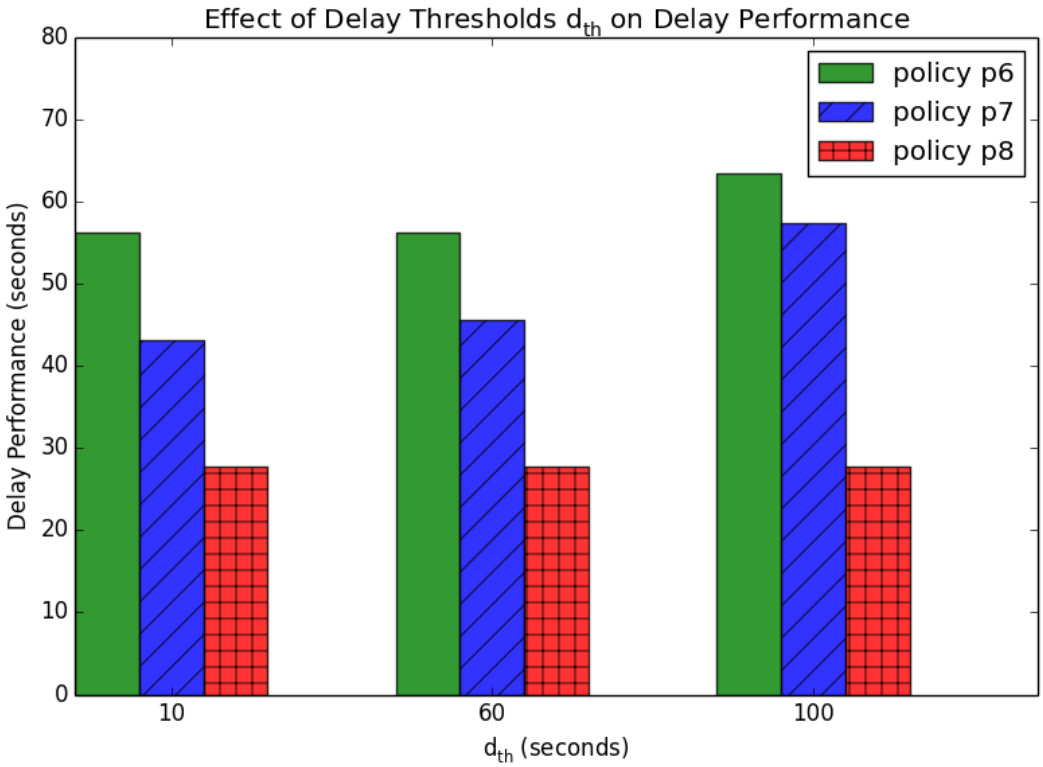

Figure 6.15 Effect of d_{th} on Objective (O3) for traffic pattern 'burst at $T/24$'.

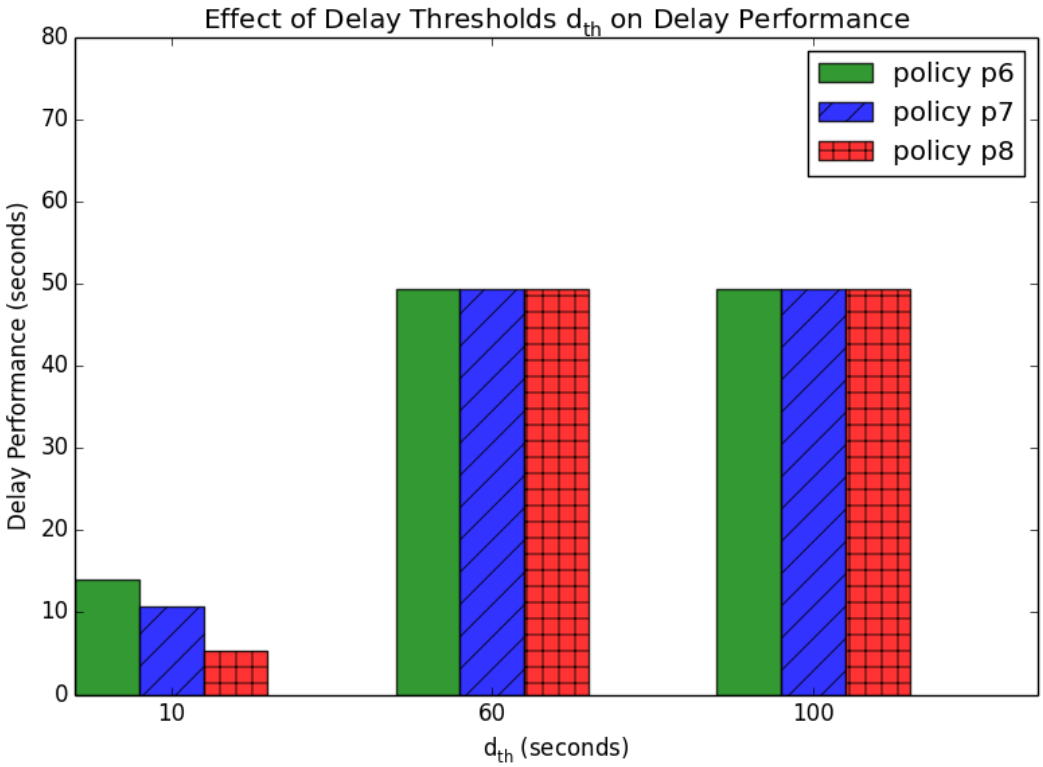

Figure 6.16 Effect of d_{th} on Objective (O3) for traffic patterns 'burst at $T/100$'.

Figure 6.20($N = 100$), we plot the tenant objective for various traffic patterns, for policy $\mathscr{P}_{fast}$. The corresponding tenant cost is shown in Figure 6.17 and Figure 6.19. Obviously for a particular N, a smaller d_{th} value gives better performance wherever the policy gets a chance to enforce the activation of a server. A high value of N allows the policy to activate more servers and thus helps to reach the threshold faster. In a burst scenario, the fast scaling approach in favour of tenant, coupled with small threshold values may cause unnecessary additional cost to the provider. As shown in Figures 6.17 and 6.19, for the traffic pattern 'burst at every T/100' and $d_{th} = 10$, the policy incurs a cost of 400 server hours ($N = 100$) and 150 server hours ($N = 10$). The average tenant performance in case of $N = 100$ comes out as 3 units which is below the $d_{th} = 10$, as the policy scaled more servers than required in the beginning to get to the chosen value of d_{th} quickly.

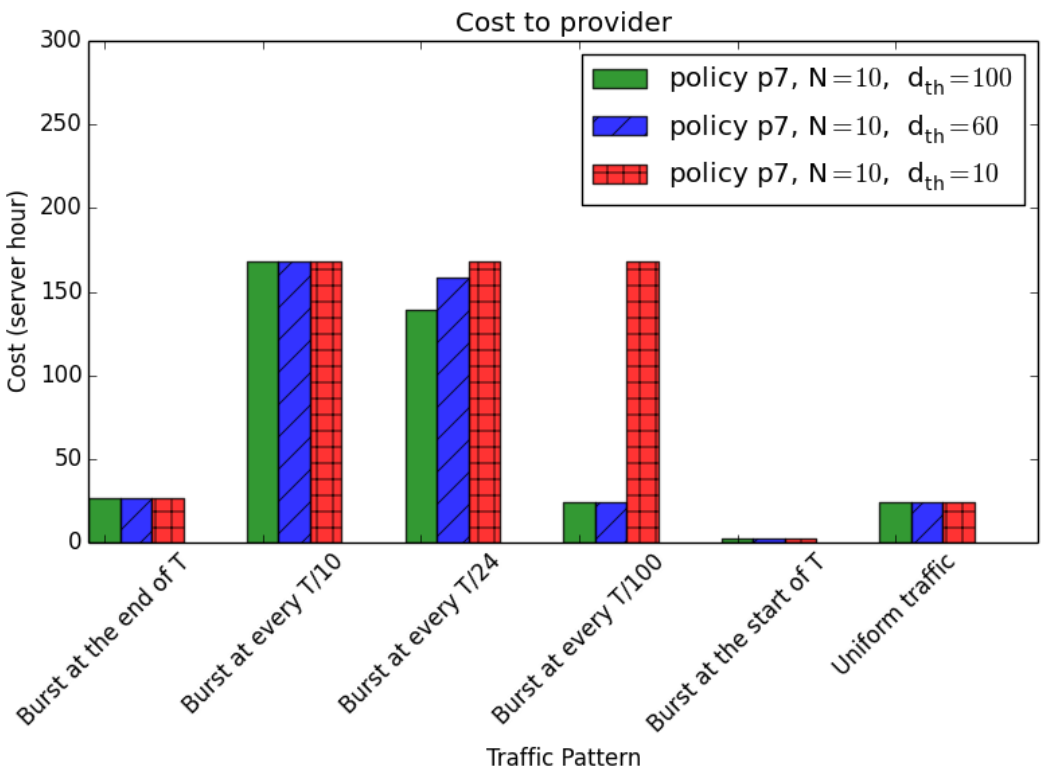

Figure 6.17 Effect of d_{th} on Objective (O2) for different traffic patterns, policy p_7.

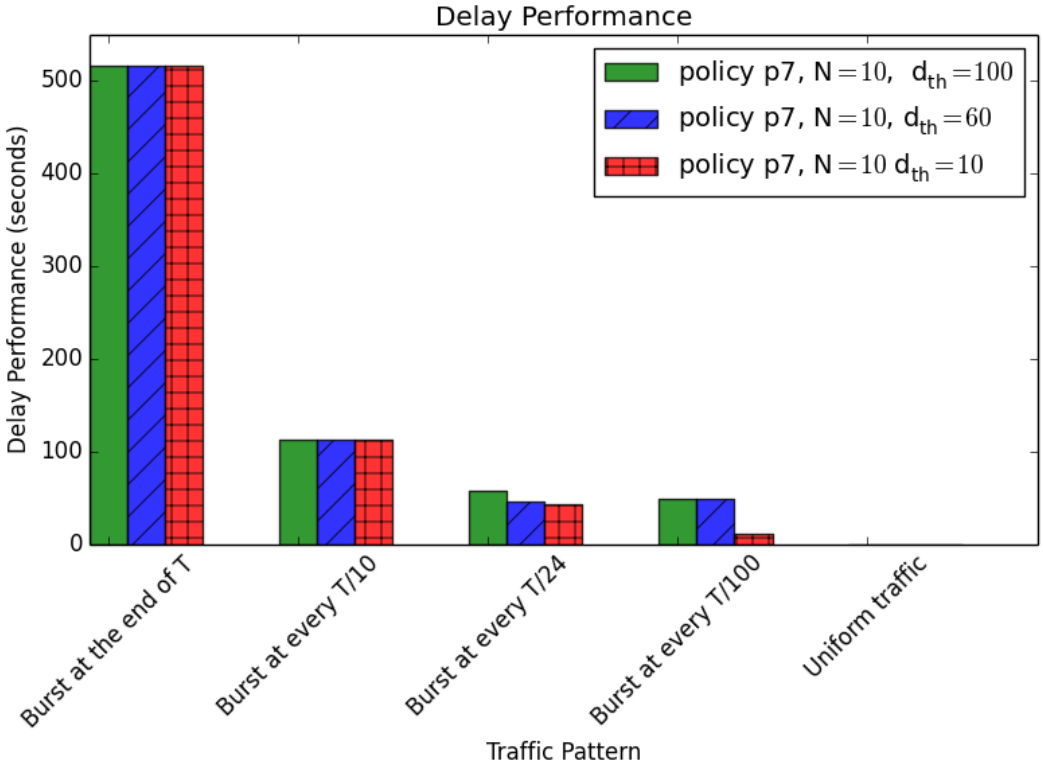

Figure 6.18 Effect of d_{th} on Objective (O3) for different traffic patterns, policy p_7.

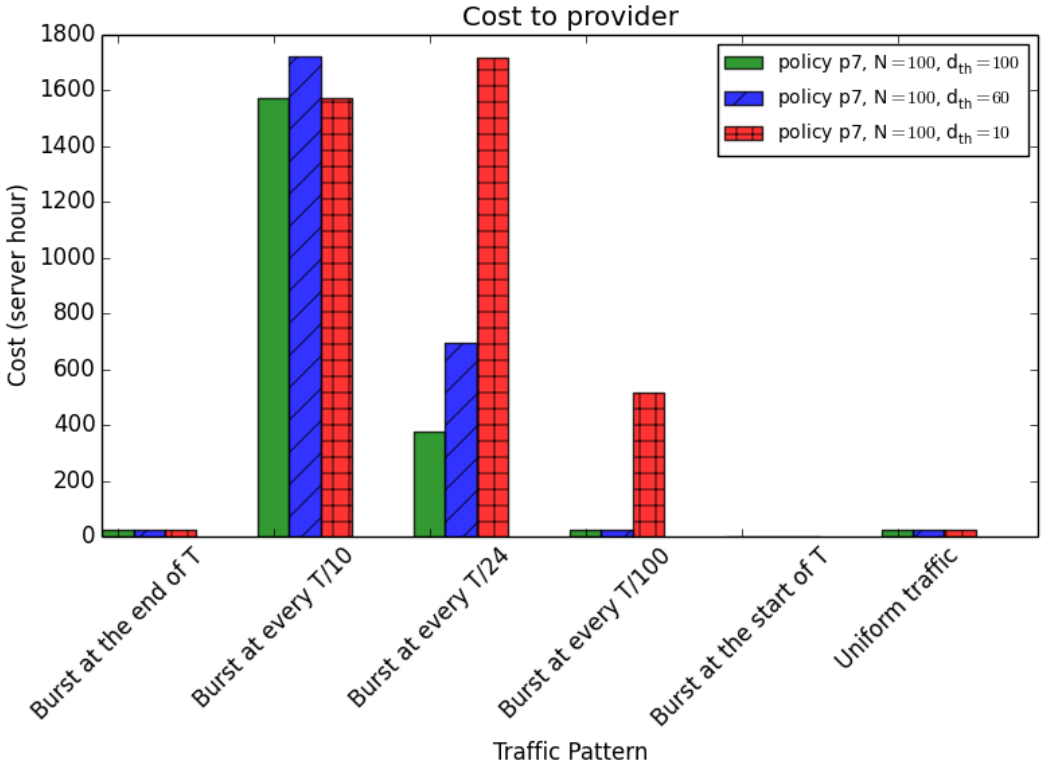

Figure 6.19 Effect of d_{th} on Objective (O2) for different traffic patterns, policy p_7.

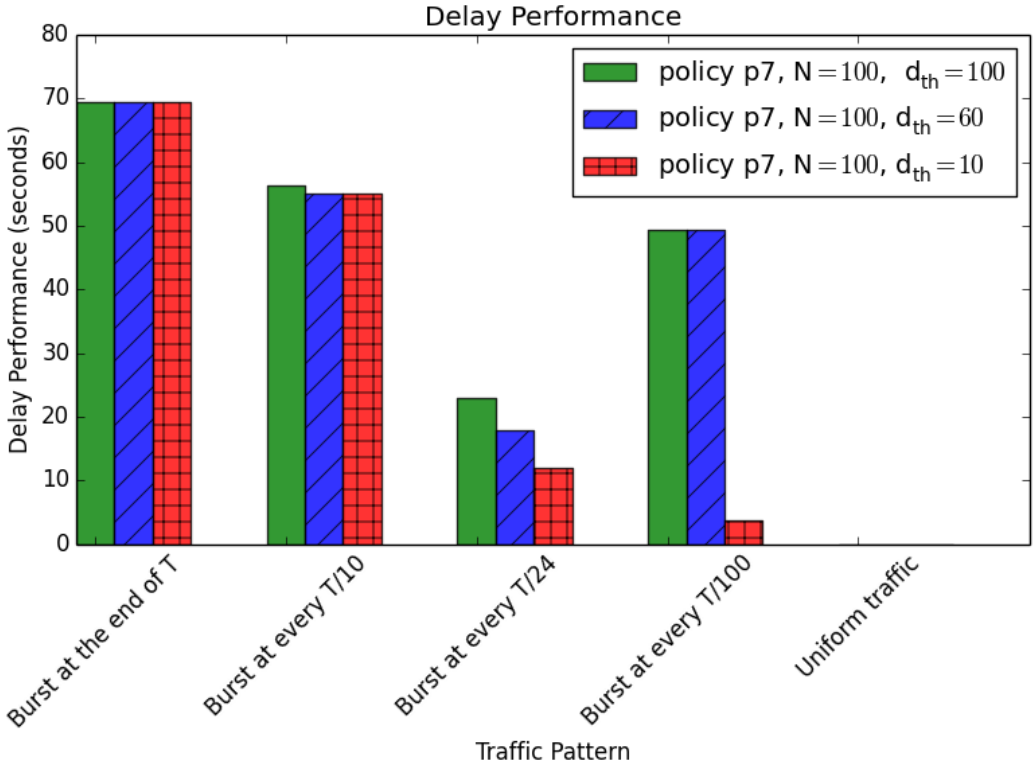

Figure 6.20 Effect of d_{th} on Objective (O3) for different traffic patterns, policy p_7.

6.5.5 Overhead and Tradeoff Analysis

6.5.5.1 Configuration complexity

In Table 6.1, the metric for configuration complexity indicates how difficult it is to "tune" the controller parameters, if any.

Table 6.1 Configuration complexity of various policies.

Policy	Type, favors	Configuration complexity
p_1	static, tenant	Zero
p_2	static, provider	Zero
p_3	static, neutral	Zero
p_4	static, neutral	Low, choice of period
p_5	static, neutral	Zero
p_6	Dynamic, tenant	Large, choice of period, d_{th}
p_7	Dynamic, tenant	Large, choice of period, d_{th}
p_8	Dynamic, tenant	Large, choice of d_{th}
p_9	Dynamic, provider	Zero

6.5.5.2 Implementation overheads

This overhead takes into account the amount of feedback measurements needed by the controller that calculates the policy decisions (see Table 6.2) as well as requirements for (a) bandwidth to send the system feedback (see Table 6.3), (b) CPU time to process this feedback (see Table 6.4), and, (c) memory to store the feedback and decisions (see Table 6.5).

6.6 Conclusion

In this chapter, we addressed the problem of (server) resource provisioning in a cloud environment with a cluster of service instances. The availability of more than one servers in this environment and the need to satisfy competing tenant and provider requirements makes the problem substantially more challenging than the one we addressed in Chapter 4. The main control is now scaling of server resources. Our main contribution in this chapter is the systematic design of scaling policies that are proven to meet the SLA and provide effective knobs to tradeoff between the two competing tenant and provider requirements.

Table 6.2 Measurement overhead of various policies.

Policy	**Type, favors**	**Measurement overhead**
p_1	static, tenant	Zero
p_2	static, provider	Zero
p_3	static, neutral	Zero
p_4	static, neutral	Zero
p_5	static, neutral	Zero
p_6	Dynamic, tenant	Large, $3X$ measurements needed for $Del(t_k^d)$
p_7	Dynamic, tenant	Large, $3X$ measurements needed for $Del(t_k^d)$
p_8	Dynamic, tenant	Large, $3X$ measurements needed for $Del(t_k^d)$
p_9	Dynamic, provider	Large, X measurements needed for t_X

Table 6.3 Bandwidth overhead of various policies.

Policy	**Type, favors**	**Bandwidth overhead**
p_1	static, tenant	Zero
p_2	static, provider	Zero
p_3	static, neutral	Zero
p_4	static, neutral	Zero
p_5	static, neutral	Zero
p_6	Dynamic, tenant	Large, $3X$ measurements needed for $Del(t_k^d)$
p_7	Dynamic, tenant	Large, $3X$ measurements needed for $Del(t_k^d)$
p_8	Dynamic, tenant	Large, $3X$ measurements needed for $Del(t_k^d)$
p_9	Dynamic, provider	Large, X measurements needed for t_X

What is the main conclusion? If one put a lot of weight on the overhead and implementation headaches, a credit of 10% is worth it for the provider...

Table 6.4 CPU processing overhead of various policies.

Policy	**Type, favors**	**CPU processing overhead**
p_1	static, tenant	Zero
p_2	static, provider	Zero
p_3	static, neutral	Zero
p_4	static, neutral	Zero
p_5	static, neutral	Zero
p_6	Dynamic, tenant	Large, $3X$ measurements needed for $Del(t_k^d)$
p_7	Dynamic, tenant	Large, $3X$ measurements needed for $Del(t_k^d)$
p_8	Dynamic, tenant	Large, $3X$ measurements needed for $Del(t_k^d)$
p_9	Dynamic, provider	Large, X measurements needed for t_X

Table 6.5 Memory overhead of various policies.

Policy	**Type, favors**	**Memory overhead**
p_1	static, tenant	Zero
p_2	static, provider	Zero
p_3	static, neutral	Zero
p_4	static, neutral	Zero
p_5	static, neutral	Zero
p_6	Dynamic, tenant	Large, $2X$ memory locations needed for $Del(t_k^d)$
p_7	Dynamic, tenant	Large, $2X$ memory locations needed for $Del(t_k^d)$
p_8	Dynamic, tenant	Large, $2X$ memory locations needed for $Del(t_k^d)$
p_9	Dynamic, provider	Zero, one memory location needed for t_X

CHAPTER

7

CONCLUSION AND FUTURE WORK

7.1 Thesis Conclusion

In this thesis, we have investigated a problem in the realm of hosting IoT services in a cloud computing environment. As IoT system adoption becomes more and more widespread, the implementation of IoT services (e.g., processing of the generated sensor measurements, analytics, storage, etc.) will use datacenters that will house several tenants, *each with his/her own measurement generation patterns as well as performance needs*. The current trend of formalizing the relationship between tenants and provider is through a *simplistically-stated, "provider-friendly" SLA* that ignores these traffic patterns and needs; we believe this trend will continue, at least until operational experience enables datacenter designers to offer different (e.g., more complicated, more "tenant-friendly") SLAs.

The main technical problem we addressed is *how can this SLA be guaranteed*? A simplistic SLA, offered to all tenants regardless of their needs and workload characteristics simplifies the job of the datacenter designer who is tasked with provisioning datacenter resources (CPU time only in this work) to guarantee the SLA. The provision of financial credits in the SLA, in the case the provider cannot meet the SLA objective, is a convenient, inexpensive "way-out" of designs that fall short.

We have tackled this technical problem by breaking it down to a series of three smaller problems;

we addressed them in Chapters 4, 6 and 5. In Chapters 4 and 6, we considered the *SLA in its current form* (i.e., "for $\$Y$, provider guarantees to have X messages processed in a month"); the difference between the two is the assumption of a *single* (Chapter 4) vs *multiple* (Chapter 6) instances of servers used to guarantee the SLA. In Chapter 5, we *modified the SLA* - we took away part of its statement simplicity in order to reduce the burden on the designed controls. The spirit of the new SLA is that by reducing the scope of the guarantee, it offers additional, more granular control "knobs" to the designer.

In the limited scope of this thesis, the main "control knob" at the disposal of the designer is *allocation of CPU service time*. When deciding how to provision CPU time, the fundamental technical challenge in all three problems is the lack of knowledge about traffic patterns. An additional challenge in the case of multiple servers is the *presence of two conflicting objectives*. Even though the (new or current) SLAs do not mention these objectives, it is to the best interest of the designer to come up with resource provisioning controls that will keep the tenant happy (in the sense of acceptable average delays, in this thesis) as well as the provider/boss happy (in the sense of keeping costs down, in this thesis).

The main contribution of this thesis is twofold: (a) the *introduction of the new SLA*, and, (b) the *design and evaluation of the CPU provisioning algorithms* to meet the SLAs. We have introduced and justified the need for the new SLA in Section 5.2.2. The algorithms we designed have been presented in Sections 4.2.4 for the single service instance problem/current SLA, 5.3.4 for the single service instance problem/new SLA, and 6.4 for the multiple service instance problem/current SLA breakdowns.

As the breakdown of the problem addresses more and more realistic environments and incorporates more and more realistic operational assumptions, the complexity of the designed algorithms increases significantly. The analysis of implementation overheads in Section 6.5.5 suggests nontrivial impact on administrators of a datacenter offering (IoT or not) services to multiple tenants. Our business-related conclusion from this work is that "offering true guarantees is a tough cookie for everyone in the cloud provider organization". Offering service credits saves the day...

7.2 Future Work

The work in this dissertation can be extended in three main directions.

In the first direction, additional business aspects can be explored, in three topics.

First, the *new SLA* we proposed in Chapter 5 can be studied with the *multiple service instance environment* of Chapter 6. This is a more realistic environment. The main goal would be to again design server scaling policies that meet the new SLA and respect the competing objectives of the

tenant and provider.

Second, the volume-based SLA can be extended to *include additional resources* besides CPU time, bandwidth in particular. This extension is much more challenging to address, without knowledge of traffic characteristics. SLAs that are still simplified to state but include (switch fabric) bandwidth guarantees will be needed in the future for a number of (not only) IoT applications.

Third, as operational experience from the current SLA accumulates, a "makeover of the SLA" can be considered. The parameter T in the current SLA has a dual business/technical role. It is used to denote the length of the billing cycle, over which credits for non-conformance are calculated. On the technical side, it is also used to enforce the SLA. At the expense of increasing the complexity of stating the SLA, one can define a separate parameter T' for technical use; we have done that, in a limited form, in the new SLA. A complete decoupling of the two uses would allow designs where the periods of enforcement could have, say, lengths that are adaptive to the traffic pattern. How would this affect the design of the policies is an interesting problem. What if other credit values are considered to favor the tenant, like 200% credit? This would definitely require sophisticated, dynamic control policies. The cost of implementing and operating under such policies should be carefully studied.

In the second direction, additional technical aspects can be explored, in nine topics.

First, one can *relax the affinity assumption* we made in Chapter 6. This way, new types of workloads can be studied. The difficulty in studying this extension will mainly come from the fact that an additional control knob, namely load balancing will come to play. The scaling policies that need to be designed in this environment must be synergistic with the load balancing control in effect. The SLA offered by the provider to its tenants can be the existing one or the one we proposed in Chapter 5.

The second problem addresses a practical (as well as business) issue, namely *how to handle traffic in excess of the SLA contracted amount* (i.e., more than X messages per month). The tenant may not be in complete control of the measurement generation process, especially in cases where the measurements are event-based. The simple policing/rate limiting controls may unduly drop important measurements. A solution with pricing tiers for traffic in excess of the contracted amount is a better solution for the tenant; it is also in the spirit of elastic computing and scaling policies [5]. In this direction, new SLAs must be formulated and, of course, more sophisticated scaling policies must be designed and evaluated.

The third problem is mainly one of academic curiosity. We have designed several scaling policies in Chapter 6; we have defined the "extreme ones" that are best for the objectives of the two competing players in the SLA. We have just scratched the surface, given the fact that the set of scaling policies has myriads of elements. *Additional classes of dynamic scaling policies* can always be designed,

especially if new workload patterns (for example East-West traffic) are taken into account.

The fourth problem addresses the issue of *scaling down a resource*. Throughout this thesis, we confined our attention to policies that scale resources up but never down. It is more realistic for a provider to take a resource away from a tenant, for a variety of reasons - for example, when the SLA is met in advance of the measurement or enforcement period. Design of scale-up & scale-down policies is much more challenging, for the current or any type of SLA for that matter. The main difficulty is the myriad of choices in the way scale down can be designed.

The fifth problem deals with the issue of *unknown traffic patterns*. In a number of use cases, repeating traffic patterns exist, in the measurement generation process, even if the tenant does not know them and the SLA does not describe them as a (Tspec) would do in an IP network. It would be interesting to design and evaluate controls that "learn such patterns" over several measurement periods and adapt the scheduling policies, in the spirit of the analysis in Section 6.5.4.2. In particular, we deal with d_{th}, the threshold knob, in an adaptive manner; this can be done naturally when we have a subperiod framework or across multiple measurement periods.

In the spirit of the business "makeover of the SLA" a sixth technical problem can be considered. At the expense of increasing the complexity of stating the SLA, one can use a different parameter T for measurements and enforcement; we have done that, in a limited form, in the new SLA. A complete decoupling of the two uses would allow designs where the periods of enforcement could have, say, lengths that are adaptive to the traffic pattern. How would this affect the design of the policies is an interesting problem.

The seventh problem addresses an implicit assumption that limits, in essence, the type of IoT applications under consideration. There is no mention in the SLA about processing time requirements; a message is defined as a block of 512 bytes. Implicitly, the time to process it is assumed to be constant. In practice, this time will be variable; it may be bounded, but with bounds that are probably unknown, at least initially, and definitely different among tenants. Structuring a new SLA that takes this service characteristic into account has business merit, especially if operational experience with the current SLA proves to be inadequate for real IoT application needs. Designing dynamic controls to meet it will be a fascinating problem.

The eight problem deals with some practical issues in a multitenant environment and scaling policies. When such policies are dynamic, they typically react to bursts of traffic by scaling up resources. When bursts from M tenants pile up, this reaction may consume large amounts of resources. The design of scaling policies in Chapter 6 was done with a single tenant in mind. Implementation in a multitenant environment can make use of some tricks, to reduce the needed resources. For example, "staging" of the measurement periods has the potential for such a reduction. A thorough study of such tricks has practical merits.

Finally, in the last problem, *additional metrics can be investigated*. From a general, not just IoT tenant perspective, there may be workloads for which volume-based metrics in the SLA do not make as much sense as, say average delay of responses or response throughput. Design of policies in the multiple service instance environment, with no or limited knowledge of traffic characteristics is a relevant, interesting and challenging problem. In particular, theoretical proofs that a designed policy will achieve the stated SLA objectives are of great interest, at least from the academic point of view.

BIBLIOGRAPHY

[1] Ali-Eldin, A. et al. "An adaptive hybrid elasticity controller for cloud infrastructures". *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE. 2012, pp. 204–212.

[2] Ali-Eldin, A. et al. "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control". *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*. ACM. 2012, pp. 31–40.

[3] Almquist, P. *Type of Service in the Internet Protocol Suite*. RFC 1349. RFC Editor, 1992. URL: `http://www.rfc-editor.org/rfc/rfc1349.txt`.

[4] *Amazon EC2 SLA - Amazon Web Services*. URL: `https://aws.amazon.com/ec2/sla/` (visited on 01/06/2017).

[5] *Amazon EC2 SLA - Amazon Web Services*. URL: `https://aws.amazon.com/ec2/pricing/on-demand/` (visited on 01/06/2017).

[6] *Amazon Elastic Compute Cloud (Amazon EC2)*. URL: `hhttps://aws.amazon.com/ec2/` (visited on 01/06/2017).

[7] *Amazon S3 Service Level Agreement*. URL: `https://aws.amazon.com/s3/sla/` (visited on 01/01/2017).

[8] Amendola, S. et al. "RFID technology for IoT-based personal healthcare in smart spaces". *IEEE Internet of things journal* **1**.2 (2014), pp. 144–152.

[9] *An Introductory Overview of ITIL 2011 - TSO Shop*. 2011. URL: `https://www.tsoshop.co.uk/gempdf/itSMF\_An\_Introductory\_Overview\_of\_ITI_V3.pdf`.

[10] *An overview of IoT, Recommendation ITU-T Y.2060 (06/2012)*. URL: `http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060`.

[11] *App Engine Service Level Agreement (SLA)- Google Cloud Platform*. URL: `{https://cloud.google.com/appengine/sla}` (visited on 01/01/2017).

[12] Ashton, K. "That 'Internet of things' thing". *RFID Journal* **22**.7 (2009), pp. 97–114.

[13] *AT&T Broadband - Service Level Agreement*. URL: `https://www.att.com/gen/general?pid=6622` (visited on 01/06/2017).

[14] *Auto scaling - Microsoft Azure*. URL: `https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/monitoring-overview-autoscale` (visited on 04/10/2017).

[15] Awduche, D. O. "MPLS and traffic engineering in IP networks". *Communications Magazine, IEEE* **37**.12 (1999), pp. 42–47.

[16] *AWS Auto Scaling*. URL: https://aws.amazon.com/autoscaling/ (visited on 04/01/2017).

[17] *AWS IoT Pricing*. URL: https://aws.amazon.com/iot-platform/pricing/ (visited on 01/01/2017).

[18] *Azure IoT Hub Pricing*. URL: https://azure.microsoft.com/en-us/pricing/details/iot-hub/ (visited on 01/01/2017).

[19] Bacigalupo, D. A. et al. "Resource management of enterprise cloud systems using layered queuing and historical performance models". *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE. 2010, pp. 1–8.

[20] Barrett, E. et al. "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud". *Concurrency and Computation: Practice and Experience* **25**.12 (2013), pp. 1656–1674.

[21] Beloglazov, A. & Buyya, R. "Energy efficient resource management in virtualized cloud data centers". *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing*. IEEE Computer Society. 2010, pp. 826–831.

[22] Beloglazov, A. et al. "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing". *Future generation computer systems* **28**.5 (2012), pp. 755–768.

[23] Blake, S. et al. *An Architecture for Differentiated Services*. RFC 2475. RFC Editor, 1998. URL: http://www.rfc-editor.org/rfc/rfc2475.txt.

[24] Boloor, K. et al. "Evaluation of multi-point to single-point service traffic shaping in an enterprise network". *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*. IEEE. 2009, pp. 1–6.

[25] Boloor, K. et al. "Dynamic request allocation and scheduling for context aware applications subject to a percentile response time SLA in a distributed cloud". *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE. 2010, pp. 464–472.

[26] Bonomi, F. et al. "Fog computing and its role in the internet of things". *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, pp. 13–16.

[27] Bormann, C. et al. *The Constrained Application Protocol (CoAP)*. RFC 7252. 2014. URL: https://rfc-editor.org/rfc/rfc7252.txt.

[28] Boyle, J. et al. *Applicability Statement for Traffic Engineering with MPLS*. RFC 3346. RFC Editor, 2002. URL: http://www.rfc-editor.org/rfc/rfc3346.txt.

[29] Braden, B. et al. *Integrated Services in the Internet Architecture: an Overview*. RFC 1633. RFC Editor, 1994. URL: `http://www.rfc-editor.org/rfc/rfc1633.txt`.

[30] Braden, B. et al. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. RFC 2205. RFC Editor, 1997. URL: `http://www.rfc-editor.org/rfc/rfc2205.txt`.

[31] Buyya, R. et al. "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities". *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*. Ieee. 2008, pp. 5–13.

[32] Chen, S.-Y. et al. "Intelligent home-appliance recognition over IoT cloud network". *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*. IEEE. 2013, pp. 639–643.

[33] Chieu, T. C. et al. "Dynamic scaling of web applications in a virtualized cloud computing environment". *E-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*. IEEE. 2009, pp. 281–286.

[34] Choi, Y. & Lim, Y. "Optimization Approach for Resource Allocation on Cloud Computing for IoT". *International Journal of Distributed Sensor Networks* **2016** (2016).

[35] Clark, D. "The design philosophy of the DARPA Internet protocols". *ACM SIGCOMM Computer Communication Review* **18**.4 (1988), pp. 106–114.

[36] Da Xu, L. et al. "Internet of things in industries: A survey". *IEEE Transactions on Industrial Informatics* **10**.4 (2014), pp. 2233–2243.

[37] Doukas, C. & Maglogiannis, I. "Bringing IoT and cloud computing towards pervasive healthcare". *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE. 2012, pp. 922–926.

[38] Dutreilh, X. et al. "Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow". *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*. 2011, pp. 67–74.

[39] Finden-Brown, C & Long, J. "Introducing the IBM process reference model for IT: PRM-IT sequencing the DNA of IT management". *IBM Global Services* (2005).

[40] *FitSM- Standards for lightweight Service Management in Federated IT Infrastructure*. URL: `http://fitsm.itemo.org/`.

[41] Fitzsimmons, J. A. et al. *Service management: Operations, strategy, and information technology*. McGraw-Hill New York, 2006.

[42] Fu, Y. & Vahdat, A. "Service level agreement based distributed resource allocation for streaming hosting systems". *Proc. of 7th Int. Workshop on Web Content Caching and Distribution*. Citeseer. 2002.

[43] Garg, S. K. et al. "SLA-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter". *international conference on Algorithms and architectures for parallel processing*. Springer. 2011, pp. 371–384.

[44] Geelan, J. *An overview of IoT, Recommendation ITU-T Y.2060 (06/2012)*. 2009. URL: `http://cloudcomputing.sys-con.com/node/612375/print`.

[45] Ghanbari, H. et al. "Exploring alternative approaches to implement an elasticity policy". *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE. 2011, pp. 716–723.

[46] *Google Cloud Pub/Sub Pricing*. URL: `https://cloud.google.com/pubsub/pricing` (visited on 01/01/2017).

[47] *Google Cloud Storage SLA*. URL: `https://cloud.google.com/storage/sla` (visited on 01/01/2017).

[48] Goyal, P. & Vin, H. M. "Generalized guaranteed rate scheduling algorithms: a framework". *IEEE/ACM Transactions on Networking (TON)* **5**.4 (1997), pp. 561–571.

[49] Grönroos, C. *Service management and marketing: customer management in service competition*. John Wiley & Sons, 2007.

[50] Gubbi, J. et al. "Internet of Things (IoT): A vision, architectural elements, and future directions". *Future Generation Computer Systems* **29**.7 (2013), pp. 1645–1660.

[51] Hasan, M. Z. et al. "Integrated and autonomic cloud resource scaling". *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE. 2012, pp. 1327–1334.

[52] Hassanalieragh, M. et al. "Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges". *Services Computing (SCC), 2015 IEEE International Conference on*. IEEE. 2015, pp. 285–292.

[53] *IBM Internet of Things Foundation*. URL: `https://www.ibm.com/marketplace/cloud/internet-of-things-cloud/us` (visited on 09/01/2016).

[54] *IBM Watson IoT Platform - Cloud*. URL: `https://internetofthings.ibmcloud.com/`.

[55] "IEEE Standard for Ethernet". *IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008)* (2012), pp. 1–3747.

[56] "IEEE Standard for Information technology–Local and metropolitan area networks–Specific requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements". *IEEE Std 802.11e-2005 (Amendment to IEEE Std 802.11, 1999 Edition (Reaff 2003)* (2005), pp. 1–212.

[57] Jarma, Y. et al. "Dynamic Service Contract Enforcement in Service-Oriented Networks". *Services Computing, IEEE Transactions on* **6**.1 (2013), pp. 130–142.

[58] Jiang, J. et al. "Optimal cloud resource auto-scaling for web applications". *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE. 2013, pp. 58–65.

[59] Kalyvianaki, E. et al. "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters". *Proceedings of the 6th international conference on Autonomic computing*. ACM. 2009, pp. 117–126.

[60] Katsalis, K. et al. "Dynamic CPU scheduling for QoS provisioning". *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE. 2013, pp. 630–635.

[61] Lasserre, M et al. *Framework for data center (DC) network virtualization*. Tech. rep. 2014.

[62] Liu, Y. et al. *Data center networks: Topologies, architectures and fault-tolerance characteristics*. Springer Science & Business Media, 2013.

[63] Lund, D. et al. "Worldwide and Regional Internet of Things (IoT) 2014–2020 Forecast: A Virtuous Circle of Proven Value and Demand" (2014).

[64] Martin, J. & Nilsson, A. "On service level agreements for IP networks". *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 2. IEEE. 2002, pp. 855–863.

[65] Maurer, M. et al. "Enacting SLAs in clouds using rules". *Euro-Par 2011 Parallel Processing* (2011), pp. 455–466.

[66] Mell, P. & Grance, T. "The NIST definition of cloud computing" (2011).

[67] *MPLS VPN Service Level Agreements*. URL: `https://www.sprint.com/business/resources/mpls_vpn.pdf` (visited on 01/06/2017).

[68] *MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta. 10 April 2014. OASIS Committee Specification Draft 02 / Public Review Draft 02.* URL: `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.html`.

[69] Perera, C. et al. “Sensing as a service model for smart cities supported by internet of things”. *Transactions on Emerging Telecommunications Technologies* **25**.1 (2014), pp. 81–93.

[70] Perros, H. G. *Networking Services - QoS, Signaling, Processes*. 2014.

[71] Pultorak, D. et al. “MOF 4.0: Microsoft Operations Framework 4.0”. *Zaltbommel: Van Haren Publishing* (2008).

[72] Quinn, J. B. et al. “Technology in services.” *Scientific American* **257**.6 (1987), pp. 50–58.

[73] Roy, N. et al. “Efficient autoscaling in the cloud using predictive models for workload forecasting”. *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE. 2011, pp. 500–507.

[74] *Scaling plans - Amaxon AWS*. URL: `http://docs.aws.amazon.com/autoscaling/latest/userguide/scaling\_plan.html\#scaling\_typesof` (visited on 04/10/2017).

[75] *Service Level Agreement - Verizon IP VPN Dedicated*. URL: `http://www.verizonenterprise.com/resources/terms/sla-ip-vpn-dedicated-services_en_xg.pdf` (visited on 01/06/2017).

[76] Shenker, S. et al. *Specification of Guaranteed Quality of Service*. RFC 2212. RFC Editor, 1997. URL: `http://www.rfc-editor.org/rfc/rfc2212.txt`.

[77] *SLA for APP services*. URL: `https://azure.microsoft.com/en-us/support/legal/sla/app-service/v1_4/` (visited on 01/01/2017).

[78] *SLA for Cloud Services*. URL: `https://azure.microsoft.com/en-us/support/legal/sla/cloud-services/v1_0/` (visited on 01/01/2017).

[79] Suciu, G. et al. “Smart cities built on resilient cloud computing and secure internet of things”. *Control Systems and Computer Science (CSCS), 2013 19th International Conference on*. IEEE. 2013, pp. 513–518.

[80] Tang, Q. et al. “Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach”. *IEEE Transactions on Parallel and Distributed Systems* **19**.11 (2008), pp. 1458–1472.

[81] Tesauro, G. et al. “A hybrid reinforcement learning approach to autonomic resource allocation”. *Autonomic Computing, 2006. ICAC’06. IEEE International Conference on*. IEEE. 2006, pp. 65–73.

[82] *The Internet of Things - Cisco, ISBG 2011*. URL: `http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`.

[83] Urgaonkar, B. et al. “Agile dynamic provisioning of multi-tier internet applications”. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* **3**.1 (2008), p. 1.

[84] Van, H. N. et al. “SLA-aware virtual resource management for cloud infrastructures”. *Computer and Information Technology, 2009. CIT’09. Ninth IEEE International Conference on*. Vol. 1. IEEE. 2009, pp. 357–362.

[85] Vaquero, L. M. et al. “A break in the clouds: towards a cloud definition”. *ACM SIGCOMM Computer Communication Review* **39**.1 (2008), pp. 50–55.

[86] *vCenter Server for vSphere Management: VMware*. URL: `https://www.vmware.com/products/vcenter-server.html`.

[87] Verma, D. C. “Service level agreements on IP networks”. *Proceedings of the IEEE* **92**.9 (2004), pp. 1382–1388.

[88] Vimercati, S. D. C. di et al. “Supporting Application Requirements in Cloud-based IoT Information Processing” ().

[89] *Web Hosting - DreamHost*. URL: `{hhttps://www.dreamhost.com/hosting/}` (visited on 01/01/2017).

[90] *Web Hosting - GoDaddy*. URL: `{https://www.godaddy.com/hosting/web-hosting}` (visited on 01/01/2017).

[91] Wroclawski, J. *Specification of the Controlled-Load Network Element Service*. RFC 2211. RFC Editor, 1997. URL: `http://www.rfc-editor.org/rfc/rfc2211.txt`.

[92] Wu, L. et al. “SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments”. *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*. IEEE. 2011, pp. 195–204.

[93] Yarmolenko, V. & Sakellariou, R. “An evaluation of heuristics for SLA based parallel job scheduling”. *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE. 2006, 8–pp.

[94] Zanella, A. et al. “Internet of things for smart cities”. *IEEE Internet of Things Journal* **1**.1 (2014), pp. 22–32.

[95] Zeithaml, V. A. et al. “Services marketing: Integrating customer focus across the firm” (2006).

[96] Zhang, Q. et al. “A regression-based analytic model for dynamic resource provisioning of multi-tier applications”. *Autonomic Computing, 2007. ICAC’07. Fourth International Conference on*. IEEE. 2007, pp. 27–27.