ABSTRACT

ZHANG, HUIJUN. Theoretical Analysis of SDA/SAA and CASTS Algorithms for Service Differentiation. (Under the direction of Dr. Yannis Viniotis.)

Many approaches to implement service differentiation in Service Oriented Architecture (SOA) environments have been recently considered and analyzed. The one we investigate in this thesis is based on controlling arrival rates by activating/deactivating instances of service domains. We prove analytically that the Service Activation/Deactivation Algorithm (SDA/SAA) can satisfy resource utilization SLAs and thus it is guaranteed to offer service differentiation.

© Copyright 2012 by Huijun Zhang

All Rights Reserved

Theoretical Analysis of SDA/SAA and CASTS Algorithms for Service Differentiation

by
Huijun Zhang

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Engineering

Raleigh, North Carolina

2012

APPROVED BY:

Dr. George N. Rouskas	Dr. Michael Devetsikiotis

Dr. Yannis Viniotis
Chair of Advisory Committee

DEDICATION

To my parents.

BIOGRAPHY

Huijun Zhang was born on September 28th, 1988 in Yangquan, Shanxi, China. She graduated from Zhejiang University in June 2010, with a Bachelor of Engineering degree in Electronics and Communication. Her undergraduate thesis was titled "Design of Polarization System".

Huijun has been working towards her Master of Science degree in Computer Engineering at North Carolina State University since August 2010. She has decided to change major to Networking. She is currently working on her thesis under the guidance of Dr. Yannis Viniotis. Her research interest is Service Oriented Networking.

She has two working experiences, the one is summer internship in Chinese Community Network, as a web developer; the other is a part time job in Appealing Products, Inc. as a software developer. She plans to pursue a career in industry after graduation.

ACKNOWLEDGEMENTS

I would like to express utmost gratitude to my advisor Dr. Viniotis for giving me the valuable opportunity to work on my Master's thesis under his extremely patient help. His knowledge, precise judgement, and solemn attitude gave me a great academic and life guidance. I am also grateful to him for providing me an opportunity to be a TA at NCSU.

My gratitude to Dr. George Rouskas and Dr. Michael Devetsikiotis for serving on my thesis committee, and help me with my thesis defence. Special thanks to Mursalin Habib and Keerthana Boloor for their help on understanding their work which formed the basis for my thesis.

I would like to thank my parents for their generous inspiration and unconditional help since I was born. They always support my decisions and give me positive attitude towards life and study no matter how many difficulties I am facing. Also thanks to my close friends Chengzhi Li, Wenbo Shen, Jianhua Zhang, Ying Yu, Bo Liu for their help.

TABLE OF CONTENTS

List of Figures . . . . . **vii**

Chapter 1 Introduction and Motivation **1**
1.1 Web Services and Service Oriented Architecture 1
1.1.1 What is a Service? 1
1.1.2 What is a Web Service? 1
1.1.3 Architectures and Implementations of Web Services 2
1.1.4 SOA 2
1.2 Prior Work Survey 3
1.2.1 Service Differentiation 3
1.2.2 Prior Approaches 4
1.3 Motivation and Contribution 4
1.4 Outline of the Thesis 5

Chapter 2 Overall System Architecture **6**
2.1 Service Domains 6
2.2 Gateway and WRR Scheduler 7
2.3 Appliances 7
2.4 Statistic Collection and SDA/SAA Block 8
2.5 Service Tier 8

Chapter 3 Proof of Convergence of the SDA/SAA Algorithm **9**
3.1 SDA/SAA Algorithm Description 9
3.1.1 Terminology 9
3.1.2 Algorithm Description 10
3.1.3 Assumptions and Notations for Proof 12
3.2 Preliminary Results 13
3.3 Statistical properties of the utilization sequence 16
3.4 The Main Result 21
3.5 Extended Results 22

Chapter 4 Conclusion and Future Work **23**
4.1 Summary and Conclusion 23
4.2 Future Work 23

References **25**

Appendix **26**
Appendix A CASTS Algorithm Proof 27
A.1 Problem Definition 27
A.2 Description of CASTS Algorithm and Manual Static Algorithm 27
A.2.1 CASTS Algorithm Introduction 27
A.2.2 Notations and clarifications for the proof 28

A.2.3 CASTS Algorithm 29
A.2.4 Manual Static Allocation 30
A.3 Proof 30
A.3.1 Case $B = 1, K = 1$ 30
A.3.2 Case $B \neq 1, K = 1$ 30
A.3.3 Case $B = 1, K \neq 1$ 31
A.3.4 The general case $B \neq 1, K \neq 1$ 33

LIST OF FIGURES

Figure 2.1 The abstract system model 7

Figure 3.1 λ_1 when $100\lfloor\frac{T}{100}\rfloor \leq T \leq 100\lfloor\frac{T}{100}\rfloor + 30$ 14
Figure 3.2 λ_1 when $100\lfloor\frac{T}{100}\rfloor + 30 \leq T \leq 100\lfloor\frac{T}{100}\rfloor + 100$ 15

Chapter 1

Introduction and Motivation

1.1 Web Services and Service Oriented Architecture

1.1.1 What is a Service?

Service can be defined as a component capable of performing a task. In the information and technology fields, service is a mechanism by which a consumer's need or want is satisfied according to a negotiated contract (implied or explicit) which includes service agreement, function offered and so on [10]. It includes technical support, computer networking, systems administration, and other IT services. Common Internet services, such as web hosting, e-mail, and social networking websites also fall under the scope of technology services.

Information and technology services may also include services not directly related to information technology, such as telephone and cable TV services. Other industries, such as digital photography, graphic design, and video production may also be considered technology services, since they involve modern technology.

1.1.2 What is a Web Service?

A Web service is a method of communication between two electronic devices over the web (internet). The World Wide Web Consortium (W3C) defines a "Web service" as "a software system designed to support interoperable machine-to-machine interaction over a network". It has an interface described in a machine-processable format (specifically Web Services Description Language, known by the acronym WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. [5]

How does a Web service work? The basic Web services platform is XML + HTTP. XML provides a language which can be used between different platforms and programming languages

and still can express complex messages and functions. The HTTP protocol is the most widely used Internet protocol. The main Web services platform elements include: SOAP (Simple Object Access Protocol); UDDI (Universal Description, Discovery and Integration); and, WSDL (Web Services Description Language).

Why use web services in the first place? Firstly, for their interoperability advantages: when all major platforms could access the Web using Web browsers, different platforms could interact. For these platforms to work together, Web-based applications were developed. Web-based applications are simply applications that run on the web. These are built around the Web browser standards and can be used by any browser on any platform. Secondly, Web services take web applications to next level: by using Web services, an application can publish its function or message to the rest of the world. Web services use XML to code and to decode data, and SOAP to transport it (using open protocols). With Web services, for example, an accounting department's billing system deployed on a Win2k server platform can connect with an IT supplier's UNIX server. [11]

1.1.3 Architectures and Implementations of Web Services

The three most widely deployed architectures and implementations of Web Services are RPC, REST and SOA.

Remote procedure call (RPC): RPC Web services present a distributed function (or method) call interface that is familiar to many developers. Typically, the basic unit of RPC Web services is the WSDL operation.

Representation state transfer (REST): REST attempts to describe architectures that use HTTP or similar protocols by constraining the interface to a set of well-known, standard operations (like GET, POST, PUT, DELETE for HTTP). Here, the focus is on interacting with stateless resources, rather than messages or operations. Clean URLs are tightly associated with the REST concept.

Service-oriented architecture (SOA): Web services can also be used to implement an architecture according to SOA concepts, where the basic unit of communication is a message, rather than an operation. This is often referred to as "message-oriented" services. We focus on this architecture next.

1.1.4 SOA

Service-Oriented Architecture (SOA) is a set of principles and methodologies for designing and developing software in the form of interoperable services. These services are well-defined business functionalities that are built as software components (discrete pieces of code and/or data structures) that can be reused for different purposes. SOA design principles are used during

the phases of system development and integration.

The following guiding principles define the ground rules for development, maintenance, and usage of SOA [2]:

a. reuse, granularity, modularity, composability, componentization and interoperability.

b. standards-compliance (both common and industry-specific).

c. services identification and categorization, provisioning and delivery, and monitoring and tracking.

There are three types of architectures: application, service, component. Application Architecture is the business-facing solution which consumes services from one or more providers and integrates them into the business processes. Service Architecture provides a bridge between the implementations and the consuming applications, creating a logical view of sets of services which are available for use, invoked by a common interface and management architecture. Component Architecture describes the various environments supporting the implemented applications, the business objects and their implementations.

SOA is implemented for a variety of reasons. Security: the Web Services Security specification addresses message security. This specification focuses on credential exchange, message integrity, and message confidentiality. Reliability: in a typical SOA environment, several documents are exchanged between service consumers and service providers. Delivery of messages with characteristics like once-and-only-once delivery, at-most-once delivery, duplicate message elimination, guaranteed message delivery, and acknowledgement become important in mission-critical systems using service architecture. Management: as the number of services and business processes exposed as services grow in the enterprise, a management infrastructure that lets the system administrators manage the services running in a heterogeneous environment becomes important. [7]

1.2 Prior Work Survey

1.2.1 Service Differentiation

The generic problem we study in this thesis is to manage the enterprise CPU resources to achieve target resource share for various classes of clients. This generic problem is very often referred to as service differentiation. A service level is used to define the expected performance behaviour of a deployed Web service, where the performance metrics are, for example, CPU processing power, average response time, supported throughput, service availability, etc. During deployment of a Web service, the resources of an underlying Web service container can be reconfigured (by acquiring new resources, if necessary) to provide a certain service level. Even the same Web service can be offered at different service levels to different clients by dynamically

allocating resources for execution of individual Web service requests [1].

1.2.2 Prior Approaches

We can classify prior work in the Service Differentiation space (and more specifically the CPU resource allocation space) under three generic approaches. The classification makes use of the system architecture model and terminology we introduce in the next chapter.

Approach 1: "manipulate the service process". Under this approach, client traffic enters the appliance as is and CPU scheduling (as discussed, for example, in [9]) is used as the control mechanism to differentiate client traffic. It requires per domain buffering (and typically used in the server tier of the model we will discuss in the next chapter).

Approach 2: "manipulate the arrival process". Under this approach, we control the rate at which client traffic enters the appliance. In a stable system, this effectively controls the output rate and hence the service the client traffic receives. We refer to this arrival control as activation/deactivation of traffic at the gateway. If, for example, more CPU resource is required to achieve the goal, client traffic rate gets increased. Thus we control the allocation of resources or performance class indirectly, as resource utilization or performance depends on arrival rate of that client. [6]

Approach 3: distinguish congestive and non-congestive queue. It is simple but powerful tool implemented in gateway since non-congestive flows do not cause significant delays and hence should not suffer from delays. Non Congestive Queuing (NCQ) is used as a specific scheduling discipline. It shows that (i) non - congestive data gets a much better service from the network, (ii) congestive data suffers no extra delays; instead, both fairness and efficiency are occasionally improved due to lower contention. Having more flows finishing early may lead to better resource sharing and more user satisfies. It is particularly beneficial for applications that utilize small rates and short packets. [8]

1.3 Motivation and Contribution

The work in [6] provided the first known methodology to provide service differentiation and make efficient use of appliance resources. The (SDA/SAA) algorithm proposed in [6] was shown to have good implementation advantages, but was evaluated via simulations only. In this thesis our motivation was to strengthen the commercial appeal of this algorithm by proving analytically that it possesses the capability of satisfying service guarantees.

In short, the main contribution of this thesis is the derivation of a theoretical proof that the SDA/SAA algorithm defined in [6] is capable of achieving the Service Level Agreement (SLA) under a variety of realistic assumptions. This result is stated as Theorem 1 in Section 3.4.

1.4 Outline of the Thesis

The thesis is organized as follows. In Chapter 2, we introduce the system architectural model used for implementing activation/deactivation of service domains at the gateway. We explain in sufficient detail the main components of the model, so that the mathematical model can be later formulated. In Chapter 3, we introduce the SDA/SAA Algorithm and provide the proof that it can achieve and desired CPU utilization. We outline the assumptions necessary for this proof. We conclude in Chapter 4 with a summary and a suggestion for future work.

In Appendix A, we outline CASTS, another algorithm used for service differentiation. CASTS has been proposed in [3] for service traffic shaping. It was evaluated via simulations only. We provide a theoretical proof of its properties.

Chapter 2

Overall System Architecture

In this chapter, we present a system model that abstracts the operation of the real systems. The model can be used to formally describe the SDA/SAA activation/deactivation algorithm and define the quantities of interest in stating and proving our main result.

The model was first used in [6] and is shown in Figure 2.1. It captures the operations of modern-day multi-tiered data centers or back-end offices in which large scale applications run on one or multiple servers (i.e., the service tier). These applications use the services of specialized "appliances" (also known as application controllers) which are typically deployed in a separate tier that resides in front of the service tier as Figure 2.1 suggests.

A typical request is served first in the appliance CPU and then the server CPU. Traffic into an appliance comes from the gateway device. This device collects traffic from users who may be dispersed all over the world.

We describe next some of the important elements of this architecture.

2.1 Service Domains

Service requests to be served in the system arrive at the gateway from clients, over a transport network. Different types of service requests are grouped into *Service Domains*. In a certain service domain, there are multiple service requests waiting in a first come first serve queue. We assume that there are M such service domains.

In the system model, if the queue is an infinite queue, assume that there are infinite requests waiting to be sent to an appliance and be served. The service time for a request is a finite random number which is determined by client and network, not determined by system or algorithm. If it is a finite queue, there maybe 0 or more requests waiting in the queue. No requests will be sent if the queue size is 0. Requests from same service domain may be sent to one or more appliances.

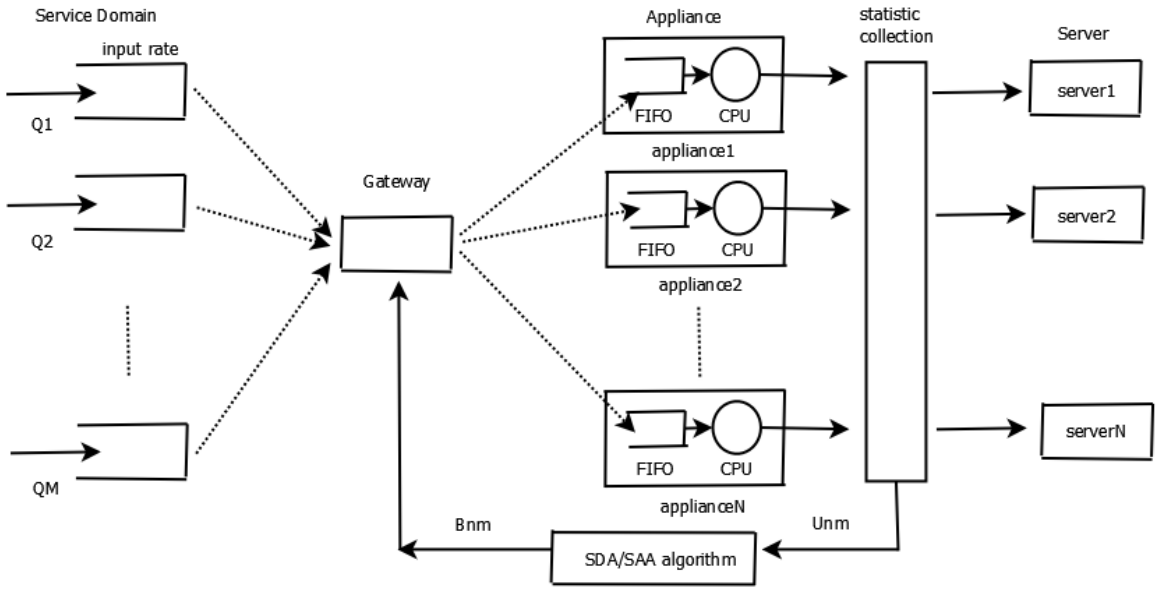

Figure 2.1: The abstract system model

2.2 Gateway and WRR Scheduler

The gateway is the first entry point of the system, where service requests from customers arrive via a transport network. Requests are placed in queues organized per service domain. Traffic from these queues is forwarded to an appliance for processing at the appliance CPU. The selection of the next packet to be forwarded to an appliance is governed by a Weighted Round-Robin (WRR) scheduler. Each service domain is assigned a weight.

The weights are not static, as is usually the case with WRR scheduler deployments. They are rather determined using feedback from the appliances. This feedback is collected by the statistics collection box as shown in Figure 2.1. As we will explain in more detail in the next chapter, this feedback contains information about CPU utilization in the appliances. The box labelled SDA/SAA algorithm processes this feedback and directs the gateway to change the WRR weights.

2.3 Appliances

Appliances are the "Middleware" for this system architecture. An appliance preprocesses incoming service requests (e.g., it performs security authentication and authorization, content based routing, mediation, XML/SOAP message processing, protocol translation) and thus removes some workload from the server tier. Each appliance has a buffer to store unprocessed service requests received from the gateway. The buffer is served in a FIFO without preemption

manner.

There are alternative architectural ways to deploy appliances, like using series connection between each other instead of parallel connection, which will reduce the cost of calculation. However, this parallel connection model can process multiple requests at the same time, and reduce average waiting time to a great extent.

2.4 Statistic Collection and SDA/SAA Block

This block (alternatively called the "provisioning agent") is responsible for collecting statistics from the appliances and calculating the WRR weights for the gateway. An increase of one unit in the weight of a service domain has been known as an "activation of an instance" of this domain; similarly, a decrease of one unit in the weight of a service domain has been known as a "deactivation of an instance" of this domain. The block is a logical entity. It may be implemented in one or more appliances or as a stand-alone or distributed application.

2.5 Service Tier

Servers process the bulk of the service requests. After finishing their preprocessing in an appliance, requests are sent to servers for final servicing. This is why, this tier is known as "server tier". Before the advent of Service Oriented Architecture in the service industry, the "server tier" used to be the only tier beyond the Gateway or load balancer.

Chapter 3

Proof of Convergence of the SDA/SAA Algorithm

In this chapter, we provide a mathematical proof that the SDA/SAA algorithm will achieve the target CPU utilizations. Our main result is Theorem 1 in Section 3.4. The result states that the sequence of utilizations achieved by the SDA/SAA algorithm will converge almost surely to the specified target CPU utilizations. In Section 3.1, we provide a summary description of the algorithm and how it is implemented in the system model described in Chapter 2. Furthermore, we state some assumptions that are necessary for the proof. In Section 3.2, we state and prove two preliminary lemmas regarding WRR scheduler properties. In Section 3.3, we study the statistical properties of the utilization sequence; they form the foundation of the convergence proof, which we present in Section 3.4. We outline and prove some extensions in the last section of this chapter.

3.1 SDA/SAA Algorithm Description

In order to describe the algorithm formally, we need to introduce some terminology first. The terminology is borrowed from [6].

3.1.1 Terminology

Provisioning Agent (PA): the PA is a logic module responsible for deciding on activation/deactivation of service domain instances in the appliance cluster. This agent can be implemented as a centralized or distributed application, residing on one or more appliances or a separate compute node. How PA collects the measured statistics from the appliance cluster is out of the scope of this thesis.

Decision Instant (T_k) is the k^{th} decision instant at which PA activates/deactivates service domain instances based on the algorithm outcome. At T_k, all the measurements collected in the time interval $(T_{k-1}, T_k]$ are evaluated; activation and deactivation of service domains are enforced. In our proofs, T_k is assumed to form a periodic sequence, for simplicity.

Target CPU percentile (P_m) is the desired percentage of CPU resources to be allocated to the m^{th} service domain.

Achieved CPU percentile ($X_m(T_k)$) is the percentage of the cluster CPU resources obtained by the m^{th} service domain until time T_k.

Down and Up Tolerances DT_m and UT_m: in order to avoid unnecessary oscillations and overhead, when the Achieved CPU percentile is "close enough" to the Target CPU percentile, i.e., when

$$P_m - DT_m < X_m(T_k) < P_m + UT_m, \tag{3.1}$$

the service domain is excluded from activation/deactivation.

Utilization Matrix (U_{nm}) is the achieved resource utilization (e.g., total CPU time used) by the m^{th} service domain in the n^{th} appliance, in the time interval $(T_{k-1}, T_k]$.

Instantiation Matrix (B_{nm}) is the number of instances of the m^{th} service domain that should be activated in the n^{th} appliance during the time interval $(T_{k-1}, T_k]$. This is the main decision variable that the PA computes.

N is the total **Number of Appliances** in the cluster.

M is the **Number of Service Domains** supported by the system.

Groups A and D denote the ranking of service domains. When service domain m is not achieving its Target CPU percentile (P_m), the PA selects it to be activated in the next decision instant in one or more appliances and thus includes it in Group A. Similarly, when service domain m is allocated more than its Target CPU percentile (P_m), the PA selects it to be deactivated in the next decision instant in one or more appliances and thus includes it in Group D.

3.1.2 Algorithm Description

At each decision instance, at time T_k, $k = 1, 2, \ldots$, we assume that measurements regarding the utilization are collected from the N appliances. More specifically, U_{nm} denotes the utilization that domain m achieved at appliance n during the time interval $(T_{k-1}, T_k]$. For simplicity of notation we drop the dependence of U_{nm} on k.

At time T_k, we can calculate $X_m(T_k)$, the actual percentile of allocated resources to each one of the M service domains up to time T_k, using the iterative equation 3.2:

$$X_m\left(T_k\right) = \frac{1}{kN}\sum_{n=1}^{N} U_{nm} + \frac{k-1}{k} X_m\left(T_{k-1}\right). \tag{3.2}$$

This equation is a recursive way of calculating the long-term time average of the CPU utilization.

The proposed algorithm uses the $X_m\left(T_k\right)$ metric to evaluate and rank performance of the domains, i.e., to check if the goal is met. Intuitively, the lower the value of $|X_m(T_k) - P_m|$ is, the "better" the performance of that particular service domain. The service domain is placed in Group A or D as follows. When

$$X_m(T_k) - P_m \geq 0, \tag{3.3}$$

the service domain meets or exceeds its target at the current time and is thus included in Group D. When

$$X_m(T_k) - P_m < 0, \tag{3.4}$$

the domain misses its target at the current time and is thus included in Group A.

Deactivation/activation algorithm is a mechanism to decide on how to deactivate (resp. activate) instances of service domains in Group D (resp. group A). The SDA/SAA algorithm is defined as follows:

(SDA) Deactivate one instance of every service domain in Group D in appliances which run service domains in Group A to free up CPU cycles utilized by domains in Group A.

(SAA) Using the instantiation matrix B_{nm}, activate one instance of every service domain in Group A, in appliances which run service domains of Group A.

There are three restrictions: Restricting the algorithm to activate equal or less than the number of deactivations it carried out in the same execution; Restricting the algorithm not to deactivate service domain instances if that is the last activated instances of a service domain in an appliance; Restricting the algorithm to activate equal or less than the number of accumulated credit, accumulated by deactivations.

At the end of each time interval, we deactivate instances in Group D, and activate instances in Group A based on achieved utilization and target utilization. Intuitively, we expect that the actual utilization will approach the target CPU utilization, so finally that there will be no more activation/deactivation in the system, and the utilization will converge to the target CPU utilization.

Feedback: the PA passes the values of the matrix B_{nm} to the gateway.

The main contribution of this thesis is a rigorous proof that the SDA/SAA algorithm will achieve any desired, pre-specified goals P_m, under the assumptions mentioned in the next section.

3.1.3 Assumptions and Notations for Proof

We make the following assumptions:

1. The service times for a given service domain m are random variables and have (finite) expected values ES_m. S_m^k denotes a generic service time in time interval $(T_{k-1}, T_k]$; we assume that $MIN \leq S_m^k \leq MAX$, where MIN and MAX are positive real numbers.

2. For each domain, there is an infinite queue of requests inside the gateway.

We make the first assumption primarily to simplify notations in our proof. We make the second assumption to avoid the unnecessary complexities of "starvation". First, consider a single appliance for simplicity. Suppose that the rate at which requests from service domain i enter the appliance is λ_i. Then, the (maximum possible) utilization of this domain is $\lambda_i \cdot ES_i$. So, when the goal is to allocate a percent P_i of CPU time to domain i, we make the implicit assumption that, under our control, we will be able to get the rate of this domain equal to λ_i. It should be clear that, if the rate at which requests enter the router is less than λ_i, we cannot devise a control to achieve P_i. Our SDA/SAA algorithms simply try to increase or decrease the rate of the arrival process to the appliances. They can definitely decrease it, but what happens if the arrival rate is too small? No matter how we try to activate more instances, we could not reach the required rate.

The instantiation matrix is used at the router to send requests to the appliances in a Weighted Round Robin (WRR) manner, with weights dictated by the matrix B_n.

By definition the achieved CPU percentile is expressed as

$$X_m(T_k) = \frac{\sum_{i=1}^{k} \sum_{j=1}^{N} U_{jm}^i}{T_k}$$

Let

$$U_m^k = \frac{1}{N} \cdot \sum_{n=1}^{N} U_{nm}(T_k) \tag{3.5}$$

denote the actual percentile of allocated time to service domain m during time interval $(T_{k-1}, T_k]$. Since we assume that $T_k = k, k = 1, 2...$, then

$$X_m(T_k) = \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{N} U_{jm}^i = \frac{1}{k} \sum_{i=1}^{k} U_m^i$$

so that $X_m(T_k)$ is the sample average of the random variable sequence $\{U_m^i\}_1^\infty$.

The proof is structured in parts, based on the assumptions we make regarding the values of tolerances and queue sizes. For the basic proof we assume that $DT_m = UT_m = 0$ and that the queue sizes are infinite. We relax these assumptions in later sections.

3.2 Preliminary Results

In this section, we present three simple lemmas; we use them in the proof of the main result.

Lemma 1 *Consider a queueing system with a single server of capacity 1 and M queues with an infinite number of requests in each queue. The queues are served by a WRR scheduler, with weights $w_i, i = 1, \ldots, M$, where w_i is a given nonnegative integer. Assume the service times are equal to 1 for all queues. Then the output rate λ_i for queue i is given by:*

$$\lambda_i = \frac{w_i}{\sum_{i=1}^{M} w_i} \tag{3.6}$$

Proof: Let $N_j(0,T)$ denote the number of requests from queue j, serviced by the WRR scheduler in time $[0,T)$. T is integer because the requests are emitted one by one per unit time. Although from intuition, WRR scheduler processes requests from service domain 1 to service domain M, we assume that it processes requests starting from service domain j in each period, then go to other service domains. This assumption is made to simplify the notation in the proof.

$$N_j(0,T) = w_j \times k + T - \sum_{i=1}^{M} w_i \times k, if \sum_{i=1}^{M} w_i \times k \leq T \leq \sum_{i=1}^{M} w_i \times k + w_j$$

$$N_j(0,T) = w_j \times k + w_j, if \sum_{i=1}^{M} w_i \times k + w_j < T \leq \sum_{i=1}^{M} w_i \times (k+1)$$

in which $k = \lfloor \frac{T}{\sum_{i=1}^{M} w_i} \rfloor$ is an integer.

$$\frac{N_j(0,T)}{T} = 1 - \frac{\sum_{i=1}^{M} w_i - w_j}{T} \times \lfloor \frac{T}{\sum_{i=1}^{M} w_i} \rfloor, if \sum_{i=1}^{M} w_i \times k \leq T \leq \sum_{l=1}^{M} w_l \times k + w_j$$

$$\frac{N_j(0,T)}{T} = \frac{w_j}{T} \times \lfloor \frac{T}{\sum_{i=1}^{M} w_i} \rfloor, if \sum_{i=1}^{M} w_i \times k + w_j < T \leq \sum_{i=1}^{M} w_i \times (k+1)$$

when $T \to \infty$, both of the equations above converge to the limit $\frac{w_j}{\sum_{i=1}^{M} w_i}$. △

Verification with example:

We exemplify the proof above using an example. Assume $w_1 = 30, \sum_{i=1}^{M} w_i = 100, S_i = 1$. $k = \lfloor \frac{T}{100} \rfloor$. The target output rate is $\frac{w_1}{\sum_{i=1}^{M} w_i} = 0.3$.

According to proof above, $N_1(0,T)$ is expressed by the following two expressions.

$$\lambda_1 = 1 \times \frac{N_1(0,T)}{T} = 1 - \frac{70}{T}\lfloor\frac{T}{100}\rfloor, when 100\lfloor\frac{T}{100}\rfloor \leq T \leq 100\lfloor\frac{T}{100}\rfloor + 30$$

$$\lambda_1 = 1 \times \frac{N_1(0,T)}{T} = \frac{30}{T}\lfloor\frac{T}{100}\rfloor, when 100\lfloor\frac{T}{100}\rfloor + 30 \leq T \leq 100\lfloor\frac{T}{100}\rfloor + 100$$

$$lim_{T\to\infty}\lambda_1 = limT \to \infty(1 - \frac{70}{T}\lfloor\frac{T}{100}\rfloor) = 0.3, when 100\lfloor\frac{T}{100}\rfloor \leq T \leq 100\lfloor\frac{T}{100}\rfloor + 30$$

$$lim_{T\to\infty}\lambda_1 = limT \to \infty(\frac{30}{T}\lfloor\frac{T}{100}\rfloor) = 0.3, when 100\lfloor\frac{T}{100}\rfloor + 30 \leq T \leq 100\lfloor\frac{T}{100}\rfloor + 100$$

As a result, the output rate converges to 0.3 which is the target output rate. These two equations above are shown in Figures 3.1 and 3.2.

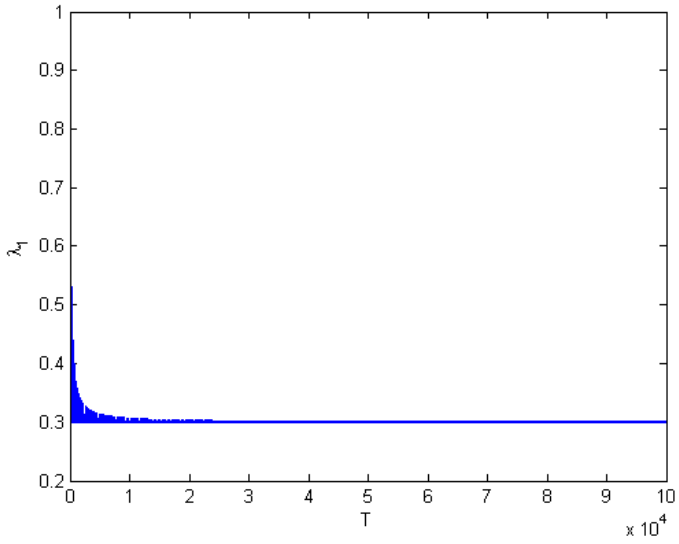

Figure 3.1: λ_1 when $100\lfloor\frac{T}{100}\rfloor \leq T \leq 100\lfloor\frac{T}{100}\rfloor + 30$

Lemma 2 *Consider a queueing system with a single server of capacity 1 and M queues with an infinite number of requests in each queue. The queues are served by a WRR scheduler, with integer, nonnegative weights $w_i, i = 1, \ldots, M$. Suppose that we are given rational constants*

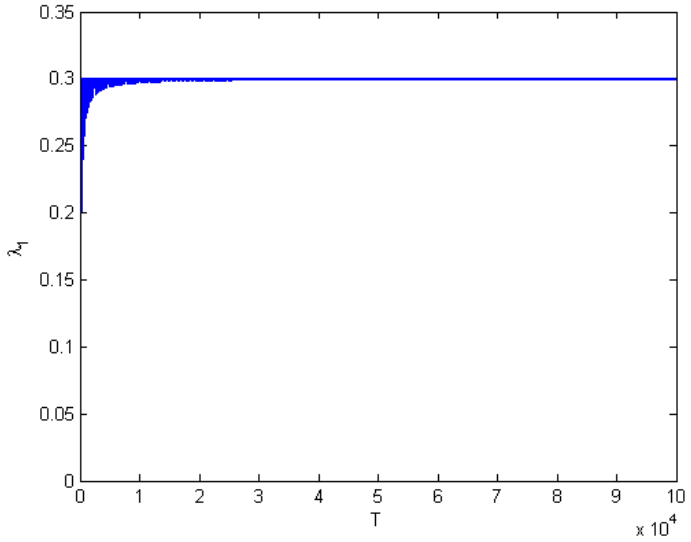

Figure 3.2: λ_1 when $100\lfloor\frac{T}{100}\rfloor + 30 \leq T \leq 100\lfloor\frac{T}{100}\rfloor + 100$

$\lambda_i, i = 1, \ldots, M$, *where*

$$\lambda_i \geq 0, \sum_{i=1}^{M} \lambda_i = 1.$$

Assume service times for all service domains are 1. Then, in order to achieve target output rate λ_i for queue i, the weights for WRR scheduler should be selected proportional to the desired output rates, i.e.,

$$w_i = L \times \lambda_i, \quad i = 1, \ldots, M, \tag{3.7}$$

where L is a constant.

Proof: According to Lemma 1, the output rate for each queue will be a rational number.

Since each λ_i is rational, we can multiply each λ_i by a (minimum possible) integer H, such that $\lambda_i' = \lambda_i \times H$ is integer, and $\sum_{i=1}^{M} \lambda_i' = H$.

According to Lemma 1, the output rate for service domain i is $\frac{w_i}{\sum_{j=1}^{M} w_j}$. Let

$$\frac{w_i}{\sum_{j=1}^{M} w_j} = \lambda_i = \frac{\lambda_i'}{H}$$

Assume $\sum_{j=1}^{M} w_j = X$, where X is an integer. We have

$$w_i = \frac{X}{H} \times \lambda_i'$$

λ_i is an integer, in order to make w_i and integer, we need to choose X such that the fraction K

$$K = \frac{X}{H}$$

is an integer. As a result,

$$w_i = K \times \lambda_i' = K \times H \times \lambda_i = L \times \lambda_i$$

where L is an integer. △

Lemma 3 *Consider two functions f, g and random variables v_1, v_2, v_3, v_4. Let $A = f(v_1, v_2), B = g(v_3, v_4)$. When v_1, v_3 take constant values, and v_2 and v_4 are independent, the random variables A and B are independent.*

Proof: Consider the event

$$P(A \leq a, B \leq b) = P(f(v_1, v_2) \leq a, g(v_3, v_4) \leq b)$$

Given the condition $v_1 = c, v_3 = d$, the event $f(c, v_2) \leq a$ only depends on the random variable v_2, the event $g(d, v_4) \leq b$ only depends on the random variable v_4. Therefore,

$$P(f(c, v_2) \leq a, g(d, v_4) \leq b) = P(f(c, v_2) \leq a) \times P(g(d, v_4) \leq b) = P(A \leq a)P(B \leq b)$$

According to the definition of independence, A and B are independent. △

3.3 Statistical properties of the utilization sequence

For simplicity of notation, in cases where there is no risk of confusion, we drop the dependence of the four variables (i.e., $U(T_k), W(T_k), A(T_k), D(T_k)$) on T_k.

Let $\{U_m^k\}_{k=1}^{\infty}$ denote the sequence of domain m utilizations achieved by the SDA/SAA algorithm, as defined in Equation 3.5. In this section, we establish the statistical properties of this sequence. More specifically, we show that $\{U_m^k\}_{k=1}^{\infty}$ is a sequence of ϕ-mixing (see Lemma 5) and identically distributed (see Lemma 6) random variables.

For a random variable sequence $\{U_m^k\}$, let $F_i^j \triangleq \sigma(U_m^k : i \leq k \leq j)$ be the σ-algebra generated. Also, let

$$\phi(r) \triangleq \sup_{\{n, A \in F_1^n, P(A) > 0\}} \sup_{\{B \in F_{n+r}^{\infty}\}} |P(B|A) - P(B)|$$

The sets A belong to the σ-algebra F_1^n after fixing n and the sets B belong to the σ-algebra F_{n+r}^{∞} for some $r > n$.

Definition: The sequence $\{U_m^k\}$ is ϕ-mixing if $\phi(r) \to 0$ as $r \to \infty$.

The definition says that as $r \to \infty$, the supremum over all n, A and B of the term $|P(B|A) - P(B)|$ should go to zero. Loosely speaking, the random variables in a ϕ-mixing sequence are asymptotically independent.

Lemma 4 *Let x be a fixed positive integer. Then there exists an integer $y < \infty$ such that U_m^x is independent of $\{U_m^k\}_{x+y}^{\infty}$.*

Proof: The proof is presented in 7 steps. According to the algorithm summary, U_m^k is a function of the instantiation matrix W^{k-1}, queue size matrix Q^{k-1} of each server from each service domain at time T_{k-1}, the order O^{k-1} of jobs from different service domains in the queues, and a matrix of service times of jobs in different service domains $\{\{S_{ij}\}_{i=1}^{\infty}\}_{j=1}^{M}$. Recall that the service times are iid random variables. We can denote

$$U_m^k = f_1(W^{k-1}, Q^{k-1}, O^{k-1}, \{\{S_{ij}\}_{i=1}^{\infty}\}_{j=1}^{M})$$

where the function f_1 has 4 arguments, W^{k-1} is an $N \times M$ matrix, with finite values in each row and column $0 \leq W_{mn}^{k-1} < \infty$, Q^{k-1} is an $N \times M$ matrix, with finite values in each row and column, order of jobs, and service times. To simplify notation a little, denote $a = W^{k-1}$, $b = Q^{k-1}$ and $c = O^{k-1}$; assume that the scheduler sends jobs to each server in the order from service domain 1 to service domain M. Recall that the servers' queues are served in a FIFO order. The domain of f_1 is the set $[0, \infty) \times [0, \infty) \times ...[0, \infty)$; its range is the set $[0, 1]$. We have,

$$U_m^k = f_1(a, b, c, \{\{S_{ij}\}_{i=1}^{\infty}\}_{j=1}^{M})$$

Step1: Assume that $X_m(T_x) \geq P_m$; we will show by contradictions that there always exists a finite y^1 which is independent of x so that $X_m(T_{x+y^1}) \geq P_m$.

If $X_m(T_x) \geq P_m$, but after y^1 periods, $X_m(T_{x+y^1}) < P_m$, service domain m will be placed in group A. The SDA/SAA algorithm will activate instances of this domain until the condition $X_m(T_{x+y'^1}) \geq P_m$ is satisfied. Let $y'^1 \to y^1$, so there should always exist a y so that $X_m(T_{x+y^1}) \geq P_m$. So service domain m at T_x and T_{x+y^1} will be in the same activation/deactivation group.

There are M service domains; the value of y^1 depends actually on the service domain m. Find y^1 so that all m service domains at T_x and T_{x+y^1} will be in the same activation/deactivation groups. Note that this value (which for simplicity of notation we also call y^1) only depends on $X_m(T_k)$, which depends on $\frac{1}{k}\sum_{i=1}^{k} U_m^i$, so y^1 is independent of x.

Step2: There always exists a finite y^2 so that $W^x = W^{x+y^2}$.

W is an $M \times N$ matrix, every element of it is an integer which shows how many instances should be activated in a certain server. This integer is finite, as we prove next.

Assume that at time T_x, we have $X_m(T_x) < P_m$.

Then service domain m will be in the activation group, and in the following time interval, WRR scheduler will send more traffic from this service domain to the servers. If in subsequent time intervals the utilization $X_m(T_{x+1})$ is still less than P_m, then WRR scheduler will send more and more traffic from this service domain to the servers.

Since service time $S_i^x \leq MAX$, in a finite time this traffic that came before T_x will leave the queues, and utilization for service domain m will start to increase, until at some time instant T_{x+y^2}, we have $X_m(T_{x+y^2}) > P_m$. At this point, the value of B_m will start to decrease because service domain m is in the deactivation group. So, the elements of W cannot increase to infinity, they have a finite upper bound G. Formally, for any time instant k, we must have $0 \leq W_{nm}^k \leq G < \infty$.

Assume that every server cannot have more than G instances activated, so W can only have finite values; there always exist infinite numbers of y^2 such that $y_1^2, y_2^2, y_3^2, \ldots$ and $W^x = W^{x+y_i^2}$. y^2 depends only on W, G but not x.

Step3: There always exists a y^3 so that $Q^x = Q^{x+y^3}$.

The queue sizes at time T_{k-1} depend on the queue sizes at T_{k-2}, the instantiation matrix, the service times of jobs in the queues (in general on $\{\{S_{ij}\}_{i=1}^{\infty}\}_{j=1}^{M}$), and O^{k-2}, the order of jobs from various service domains at T_{k-2}. More formally, we have

$$Q^{k-1} = f_2(W^{k-1}, Q^{k-2}, O^{k-2}, \{\{S_{ij}\}_{i=1}^{\infty}\}_{j=1}^{M})$$

where the function f_2 has 4 arguments, similar to function f_1. The domain of f_2 is $[0,\infty) \times [0,\infty) \times \ldots [0,\infty)$ and its range is $[0,\infty) \times [0,\infty) \times \ldots [0,\infty)$.

Assume the time to send jobs from the gateway to servers is negligible. In a stable system, if we don't want the queue sizes go to infinity, the input rate should be less than or equal to output rate. During time interval T, the maximum number of jobs can be served in an server is $\frac{T}{MIN}$ where MIN denotes the minimum value that a service time can take.

So Q can only have finite kinds of value, so there always exists infinitely many numbers $y_1^3, y_2^3, y_3^3, \ldots$ and $W^x = W^{x+y_i^3}$. y^3 depends only on W, G but not x.

Step4: There always exists a y^4 so that the order of jobs from different service domains at time x are the same as that at $x + y^4$.

The order of jobs from different service domains depends on how many jobs left for each service domain in each server, and how many jobs from each service domain have been sent during this time interval. We assume that in each server, the jobs from different service domains are sent in the order of $SD_1, SD_2, \ldots, SD_M$, and the server will serve them in a FIFO order. So in two different time intervals, if the number of jobs from different service domains are the same, the order of serving them will be the same. In other words, if queue sizes in these two time intervals are the same, the order of the jobs will be the same.

Step5: In the sequence $y_1^1, y_2^1, y_3^1, ..., y_1^2, y_2^2, y_3^2, ..., y_1^3, y_2^3, y_3^3, ...$, and $y_1^4, y_2^4, y_3^4, ...$ find the first value that $y_i^1 = y_j^2 = y_h^3 = y_l^4 (= y)$. Then at time instants x and $x+y$ we must have $W^x = W^{x+y}$, the same activation/deactivation group, the same queue sizes and order of jobs.

To see that this is the case, suppose to the contrary that no such y exists. Then, at least one of W, Q and ordering O are different at x and $x + y$. Order could not be different if W, Q are the same, so at least one of W, Q are different.

However, W and Q are both $N \times M$ matrices with elements bounded by G_1, G_2. So there are at most $G_1^{N\times M} \times G_2^{N\times M}$ different possible values for W and Q. The $(G_1^{N\times M} \times G_2^{N\times M} + 1)^{th}$ value, therefore, must be the same as one of the previous values.

Step6: U_m^x and U_m^{x+y} are independent. We have

$$U_m^x = f_1(a, b, c, \{\{S_{ij}\}_{i=1}^{\infty}\}_{j=1}^{M})$$

$$U_m^{x+y} = f_1(a, b, c, \{\{S_{ij}\}_{i=1}^{\infty}\}_{j=1}^{M})$$

All of their arguments are either the same or independent, according to *Lemma 3* they are independent. Furthermore, y is independent of x.

Step7: The proof in the previous step involved two time instants x and $x + y$ only; for this step, we need to prove that U_m^x is independent of $\{U_m^k\}_{x+y}^{\infty}$. According to the SDA/SAA algorithm,

$$W^k = f_3(\{X_m(T_k)\}_{m=1}^{M}) = f_4(\{U_m^k\}_{m=1}^{M}, \{X_m(T_{k-1})\}_{m=1}^{M})$$

where f_3 has M arguments (the achieved percentiles of allocated time to each service domain). Its domain is the set $[0, 1] \times [0, 1] \times \cdots \times [0,])$ and its range is a $M \times N$ matrix with finite values.

Similarly, f_4 has $2M$ arguments which are the utilization for each service domain and previous actual percentiles of allocated time for each service domain. Its domain is $[0, 1] \times [0, 1] \times ... \times [0, 1]$ and its range is a $M \times N$ matrix with finite values. We can write

$$U_m^{x+y} = f_1(W^{x+y-1}, Q^{x+y-1}, O^{x+y-1}, \{\{S_{ij}\}_{i=1}^{\infty}\}_{j=1}^{M})$$

$$U_m^{x+y+1} = f_1(W^{x+y}, Q^{x+y}, Q^{x+y}, \{\{S_{ij}\}_{i=1}^{\infty}\}_{j=1}^{M})$$

$$W^{x+y-1} = f_3(\{X_m(T_{x+y-1})\}_{m=1}^{M})$$

$$W^{x+y} = f_4(\{U_m^{x+y}\}_{m=1}^{M}, \{X_m(T_{x+y-1})\}_{m=1}^{M})$$

which only depends on random variables defined at the previous time instant.

As a result, U_m^{x+y+1} only depends on U_m^{x+y}, which is independent on U_m^x. Hence $\{U_m^k\}_{x+y}^{\infty}$ is independent of U_m^x. This concludes the proof of the lemma. △

Lemma 5 *The sequence of utilizations* $\{U_m^i\}_1^\infty$ *is* ϕ*-mixing. Moreover,* $\sum_{i=1}^\infty \phi(i)^{\cdot 5} < \infty$.

Proof: Choose any $n < \infty$ and choose a set $A \in F_1^n$ such that $P(A) > 0$. Then, by Lemma 4, there exists $r^* < \infty$ such that $\forall B \in F_{n+r}^\infty$, $|P(B|A) - P(B)| = 0$, $\forall r > r^*$. For clarification, n can be viewed as x in 4, and r^* can be viewed as y in 4. As a result,

$$\phi(r) = \sup_{\{n, A \in F_1^n, P(A)>0\}} \sup_{\{B \in F_{n+r}^\infty\}} |P(B|A) - P(B)| = 0$$

and thus $\forall r \geq r^*, \phi(r) = 0$. Furthermore, we have that $\sum_{i=1}^\infty \phi(i)^{\cdot 5} < \infty$, since $\phi(r) = 0$ when $r > r^*$. The sum has only only a finite number of nonzero terms. $\triangle$

Lemma 6 *Consider a given domain* m*. The random variables* U_m^i *are identically distributed. Moreover, we have*

$$EU_m^i = P_m, \tag{3.8}$$

i.e., the expected value of U_m^i *equals the target CPU percentile.*

Proof: Let $F_m^i(u)$ denote the distribution function of U_m^i for $i = 1, 2, \ldots$ Let $\triangle$ be the set of all initial conditions at the beginning of the time slots. $\omega \in \triangle$ stands for the initial condition like initial W^{p-1}, W^{q-1}, and Q^{p-1}, Q^{q-1} and the orders O^{p-1}, O^{q-1} in which the jobs from different service domains will be served.

For any $\omega \in \triangle$, U_m^p, U_m^q are any two random variable of CPU utilization. Assume that $W^{p-1} = W^{q-1}$, $Q^{p-1} = Q^{q-1}$ and $O^{p-1} = O^{q-1}$. $F_p(u|\omega)$ is identically distributed to $F_q(u|\omega)$, because by a previous definition,

$$U_m^k = f_1(a, b, c, \{\{S_{ij}\}_{i=1}^\infty\}_{j=1}^M)$$

in which the service times are iid random variables. The process of calculating U_m^p and U_m^q is the same (i.e., does not depend on p or q). As a result, for every $u \in R$, we have

$$F_p(u) = \int_\triangle F_p(u|\omega) P(d\omega) = \int_\triangle F_q(u|\omega) P(d\omega) = F_q(u)$$

and thus U_m^i's are identically distributed.

We next show that $EU_m^i = P_m$. Since

$$c_m = E[U_m^i] = \lim_{x\to\infty} \frac{1}{x} \sum_{i=1}^x U_m^i = \lim_{x\to\infty} X_m(T_x)$$

it suffices to show that

$$\lim_{x\to\infty} X_m(T_x) = P_m$$

The proof is by contradiction. Assume that the sample average converges to a value not equal to P_m, i.e., $\lim_{x\to\infty} X_m(T_x) \neq P_m$. Assume that for a positive real number ϵ, we have

$$\lim_{x\to\infty} X_m(T_x) = P_m + \epsilon$$

If at a certain time instant $X_m(T_k) = P_m + \epsilon$, the WRR scheduler will send less traffic from service domain m to the servers, so that after a finite time, when the traffic in the queues is served, the utilization of service domain m will decrease, so that $X_m(T_k)$ will also decrease, until eventually we have that $X_m(T_k) < P_m$.

A similar contradiction can be demonstrated with a negative ϵ, completing the proof of the lemma. △

3.4 The Main Result

Our main result is Theorem 2. It states that the SDA/SAA algorithm achieves any desired percentile utilizations. The result is a direct consequence of the following proposition from [4].

Proposition 1 *Let $\{Z_n\}$ be an identically distributed and ϕ-mixing sequence of random variables with $E[Z_n] = c$ and $\sum_{i=1}^{\infty} \phi(i)^{.5} < \infty$, Let $S_n = \sum_{i=1}^{n} Z_i$. Then*

$$\frac{S_n}{n} \to_{n\to\infty} c, \quad a.s. \tag{3.9}$$

Equation 3.9 in the proposition extends the almost sure (a.s.) convergence result of the Strong Law of Large Numbers to sequences of dependent random variables.

Theorem 1 *Let $P_m, \quad m = 1, 2, \ldots, M$, denote arbitrary target CPU utilizations, where $0 \leq P_m \leq 1$ and $\sum_m P_m = 1$. Let $X_m(T_k)$ denote the utilization achieved by the SDA/SAA algorithm up until time T_k. Then $X_m(T_k)$ converges almost surely to P_m:*

$$X_m(T_k) \to_{T_k\to\infty} P_m, \quad a.s., \quad m = 1, 2, \ldots, M. \tag{3.10}$$

Proof: Lemma 6 proves that the U_m^i random variables are identically distributed with $E[U_m] = P_m$; Lemma 5 proves that $\{U_m^i\}_{i=1}^{\infty}$ is ϕ-mixing, with $\sum_{i=1}^{\infty} \phi(i)^{.5} < \infty$; $X_m(T_k)$ is the sample average of random variable sequence $\{U_m^i\}_{i=1}^{\infty}$. As a result, according to Proposition 1, $X_m(T_k)$ converges almost surely to P_m. △

3.5 Extended Results

In this extension, we consider up and down tolerances that may have nonzero values.

Theorem 2 *Let* $P_m, \quad m = 1, 2, \ldots, M$, *denote arbitrary target CPU utilizations, where* $0 \leq P_m \leq 1$ *and* $\sum_m P_m = 1$. *Let* $X_m(T_k)$ *denote the utilization achieved by the SDA/SAA algorithm up until time* T_k. *Then*

$$P_m - DT < \lim_{k \to \infty} X_m(T_k) < P_m + UT \tag{3.11}$$

Proof: There are 4 possible cases for the limiting value of $X_m(T_k)$:

(a) $X_m(T_k)$ converges to a constant c_m, where $c_m \in [P_m - DT, P_m + UT]$.

(b) $X_m(T_k)$ converges to a random variable P'_m, where $P'_m \in [P_m - DT, P_m + UT]$.

(c) $X_m(T_k)$ doesn't converge; instead it oscillates between $[P_m - DT, P_m + UT]$.

(d) $X_m(T_k)$ doesn't converge; instead it oscillates outside $[P_m - DT, P_m + UT]$.

Case (a) is not possible because of this counter-example: if $[P_m - DT, P_m + UT] = [0\%, 100\%]$, then utilization will only depend on the initial value of B, which is a random variable, and service times which are iid random variables. Utilization can not converge to a constant.

Case (d) is not possible either; since

$$X_m(T_k) = \frac{1}{kN} \sum_{n=1}^{N} U_{nm} + \frac{k-1}{k} X_m(T_{k-1})$$

we can easily see that $X_m(T_k)$ cannot jump from a value greater than $P_m + UT$ to a value less than $P_m - DT$. Indeed, as $k \to \infty$, $\frac{1}{kN} \sum_{n=1}^{N} U_{nm} \to 0$, and $\frac{k-1}{k} \to 1$. If $X_m(T_{k-1}) > P_m + UT$, then $\frac{1}{kN} \sum_{n=1}^{N} U_{nm}$ needs to be larger than $UT + DT$, which is not possible.

Moreover, note that $X_m(T_k)$ cannot always stay outside $[P_m - DT, P_m + UT]$. If at a certain time we have $X_m(T_k) = P_m + UT + \epsilon$, the WRR scheduler will send less instances from service domain m to the appliances, so that after a finite time, when the requests of the domain are served, the utilization of service domain m will decrease and $X_m(T_l)$ will also decrease, until $X_m(T_l) < P_m + UT_m$.

As a result, the only possible cases are either (b) or (c); in both, the utilizations reach their target. $\triangle$

Chapter 4

Conclusion and Future Work

4.1 Summary and Conclusion

In this thesis, we have studied a problem in the space of service differentiation in Service Oriented Architecture environments. In such environments, service classes (service domains) compete for a share of available CPU resources. The allocation of such resources is often governed by Service Level Agreements (SLA). The specific SLA we have considered calls for allocating a given percentile P_m of CPU time to service domain m. The SDA/SAA algorithm has been proposed in the past as an efficient, feedback-based method to guarantee this SLA. The main contribution of this thesis is a theoretical proof that the SDA/SAA algorithm is capable of achieving the SLA under a variety of realistic assumptions. This result was stated as Theorem 1 in Section 3.4. The proof utilizes the theory of ϕ-mixing random processes to study the convergence properties of the sequences of service domain utilizations. These processes provide a convenient model for capturing the dynamics of dynamic control algorithms which rely on processing feedback in a history- (but not time-)dependent fashion. When the tolerance parameters allowed in the SDA/SAA algorithm are equal to zero ($UT = DT = 0$) and the buffers are assumed saturated with traffic (i.e., infinite queue sizes), the utilizations converge almost surely to the target CPU utilizations specified in the SLA. When a tolerance parameter is not equal to zero, it is not known whether the utilizations oscillate or converge; however, the algorithm will guarantee that they remain in the range $[P_m - DT, P_m + UT]$ (as it would have been expected).

4.2 Future Work

The case of unsaturated buffers presents an interesting case for future work. Suppose that arrivals to service domain m occur at a rate λ_m. There are two possibilities to be considered in this case, for a given desired percentile P_m: "enough traffic" and "insufficient" traffic. The latter

case is interesting for future work; it is formally characterized by the condition $\lambda_m \times ES_m < P_m - DT$ (DT may or may not be 0). In this case, according to the SDA/SAA algorithm description, the WRR scheduler may not be able to send enough jobs to appliances from service domain m. The CPU utilization will never reach the targeted goal. This could lead to system instability. The SDA/SAA algorithm must be modified to handle this case; note that, in its present form, the algorithm does not make use of the arrival rates. The easiest modification would be to incorporate such knowledge into the algorithm (or estimate the rates online).

REFERENCES

[1] Heiko Ludwiq Asit Dan and Giovanni Pacifici. Web service differentiation with service level agreements. *IBM developerWorks*, 2003.

[2] Yvonne Balzer. Improve your soa project plans. *IBM*, 2004.

[3] Keerthana Boloor. *Multi-point to Single-point Service Traffic Shaping*. PhD thesis, North Carolina State University, 03 2009.

[4] Ernst Eberlein and Murad S. Taqqu. *Dependence in probability and statistics*. Birkhauser Boston, Inc, Bostan, MA, 1986.

[5] Hugo Haas and Allen Brown. Web services glossary. *W3C Working Group Note*, 2004.

[6] Mursalin Habib. *Provisioning algorithm for service differentiation in middleware appliance clusters*. PhD thesis, North Carolina State University, 03 2009.

[7] Raghu R. Kodali. An introduction to soa. *JavaWorld*, 2005.

[8] Lafteris Mamatas and Vassilis Tsaoussidis. Differentiating services with non-congestive queuing(ncq). 2008.

[9] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1993.

[10] David Sprott and Lawrence Wilkes. Understanding service-oriented architecture. *CBDI Forum*, 2004.

[11] W3Schools. Web services tutorial. *W3 School Web Services Tutorial*, 2012.

APPENDIX

Appendix A

CASTS Algorithm Proof

In this appendix we provide a proof that CASTS Algorithm works. For more details on CASTS and the problem it solves in SOA, see [3].

A.1 Problem Definition

Service providers within an enterprise network are often governed by Client Service Contracts (CSC) that specify, among other constraints, the rate at which a particular service instance may be accessed. The main challenge is to enforce the global traffic contract by taking local actions at each appliance [3], which means the appliances locally shape the service requests to respect the global contract. The need exists for a dynamic, measurement-based service traffic shaping algorithm which respects the CSC and where credits are assigned to the middleware appliances (entry points) based on the current state of the system.

A.2 Description of CASTS Algorithm and Manual Static Algorithm

A.2.1 CASTS Algorithm Introduction

The author of [3] proposes CASTS (Credit-based Algorithm for Service Traffic Shaping), a reactive, measurement-based algorithm for the multi-point to single-point scenario enforcing the global contract by taking local actions at each appliance. The basic idea behind CASTS is to adjust the number of service requests allowed to be sent to the service host (credits allotted to the appliance) based on the number of service requests in the queues of the appliance which indicates the input rate at each appliance. CASTS also introduces the concept of subperiod enforcement. The observation interval, T, specified by the Service Access Requirements(SAR)

is slotted into K equal subperiods during which the updating of the credits take place at each appliance by exchanging their respective queue details and current credit assignment.

The CASTS algorithm is a decentralized algorithm for service traffic shaping in middleware appliances. In centralized implementations, appliances send their measurements to the central point (appliance). In decentralized ones, the middleware appliances (entry points) exchange information about their queues (an indication of their input rate) with their peer appliances every subperiod within the observation interval specified by the CSC.

A.2.2 Notations and clarifications for the proof

X is the number of requests per unit time.

T is the total number of time periods.

B is the total number of appliances.

K is the total number of subperiods.

Y is the number of incoming requests.

When in the k^{th} subperiod, the maximum number of requests the i^{th} appliance can handle is $x_i(k)$, the number of incoming requests is $Y_i(k)$, the actual number of requests that were handled is $r_i(k)$, the queue length of i^{th} appliance after k^{th} subperiod is $Q_i(k)$, and $R(k)$ is the total number of documents sent by all appliances after k^{th} subperiod.

In order to distinguish the CASTS algorithm from the static one, all variables refering to the static algorithm are denoted with a bar (for example $\bar{x}$.

When run on the appliances, the CASTS algorithm triggers the adaptation phase at the beginning of each subperiod where the credit assigned to the appliance is updated based on the current state of the system; during the remaining duration of the subinterval, the appliance is in the measurement phase, collecting the actual number of service requests sent to the host. The credits available for the appliance to use during the subperiod k is $x_i(k)$. During the measurement subperiod k, each appliance measures the actual number of documents $r_i(k)$ it was able to send to the service host. If there was not enough traffic entering the appliance, $r_i(k)$ could be less that $x_i(k)$. Otherwise, $r_i(k)$ should be equal to $x_i(k)$. Each appliance also measures the queue size, $Q_i(k)$, at the entry points. The number of requests in the queues of the appliance is an indication of the input rate at each appliance and will be used to determine the credits allotted to the appliance. In this way, credit assignment is performed under a weighted strategy.

At the end of the measurement subperiod k, each appliance broadcasts to all other appliances the values $r_i(k)$ and $Q_i(k)$; after receiving feedback from all other appliances, appliance i updates its local shaping rate $x_i(k+1)$ during the adaptation phase as shown in the next subsection.

Note that this algorithm guarantees that the SAR is respected at all times, as the total

amount of credits assigned to the appliances at each subinterval is always smaller or equal to the remaining allowed number of documents:

$$\sum_{1 \leq i \leq B} x_i(k) \leq X \times T - R(k)$$

where $R(k)$ is the total number of documents sent by all appliances during the previous sub-periods.

A.2.3 CASTS Algorithm

Input: k, (when $1 < k \leq K$) $r_j(k-1)$ and $Q_j(k-1)$, where $j = [1, .., B]$, $j \neq i$
Output: new count $x_i(k)$
if $k = 1$, then
    $x_i(k) \leftarrow \frac{X \times T}{B}$
else
    $r(k-1) = \sum_{j=1}^{B} r_j(k-1)$
    $R(k-1) = \sum_{n=1}^{k-1} r(n)$
    if $R(k-1) < X \times T$, then
        $D \leftarrow X \times T - R(k-1)$
        if $\sum_{n=1}^{B} Q_n(k-1) = 0$, then
            $x_i(k) \leftarrow \lfloor D \div B \rfloor$
        else
            $x_i(k) = \lfloor \frac{D \times Q_i(k-1)}{\sum_{n=1}^{B}(k-1)} \rfloor$
        end if
    else
        $x_i(k) \leftarrow 0$
    end if
end if
if $Y_i(k) \geq x_i(k)$, then
    $r_i(k) \leftarrow x_i(k)$
    $Q_i(k) \leftarrow Y_i(k) - x_i(k)$
else
    $r_i(k) \leftarrow Y_i(k)$
    $Q_i(k) \leftarrow 0$
end if

A.2.4 Manual Static Allocation

The allowed number of requests x_i from each of the appliances is the same

$$\bar{x_i} = \lfloor \frac{X \times T}{B} \rfloor$$

If the total period is divided into K sub-periods, then in each sub-period, the length of the sub-period will be $T \div K$, the allowed number of requests will be

$$\bar{x_i}(k) = \lfloor \frac{X \times T}{K \times B} \rfloor$$

A.3 Proof

In order to prove that the CASTS algorithm respects the SAR, we should prove that $R(k) \geq \bar{R}(k)$ (almost surely). We prove the result first for the special case of $B = 1, K = 1$; we then relax this assumption.

A.3.1 Case $B = 1, K = 1$

In this case, we can write:

$$x(1) = \lfloor \frac{X \times T}{B} \rfloor, \bar{x}(1) = \lfloor \frac{X \times T}{K \times B} \rfloor = \lfloor X \times T \rfloor$$

The actual number of requests that is handled is the minimum of $x_i(k)$ and $Y_i(k)$. In this case,

$$r(1) = \min(x(1), Y(1)) = \min(\lfloor X \times T \rfloor, Y(1)), \bar{r}(1) = \min(\bar{x}(1), Y(1)) = \min(\lfloor X \times T \rfloor, Y(1))$$

Since there is no previous sub-period, the actual number of requests that were handled in previous sub-periods are 0. Then,

$$R(1) = \sum_{n=1}^{k} r(n) = r(1) = \min(\lfloor X \times T \rfloor, Y(1)), \bar{R}(1) = \sum_{n=1}^{k} \bar{r}(n) = \bar{r}(1) = \min(\lfloor X \times T \rfloor, Y(1))$$

Therefore, $R(1) = \bar{R}(1)$ and the result holds.

A.3.2 Case $B \neq 1, K = 1$

In this case, we can write

$$x_i(1) = \lfloor \frac{X \times T}{B} \rfloor, \bar{x}_i(1) = \lfloor \frac{X \times T}{B} \rfloor$$

$$r_i(1) = \bar{r_i}(1) = \min(\lfloor \frac{X \times T}{B} \rfloor, Y_i(1))$$

$$r(1) = \sum_{i=1}^{B} r_i(1), \quad \bar{r}(1) = \sum_{i=1}^{B} \bar{r_i}(1)$$

so $r(1) = \bar{r}(1)$. As stated before, $R(1) = \bar{R}(1)$ and the result holds.

A.3.3 Case $B = 1, K \neq 1$

In this case, we use induction on the subperiod k; when $k = 1$, we can write

$$x(1) = \lfloor X \times T \rfloor, \bar{x}(1) = \lfloor \frac{X \times T}{K} \rfloor$$

Since $K \geq 2$ we have that $x(1) \geq 2 \times \bar{x}(1)$.

$$r(1) = \min(x(1), Y(1)), \bar{r}(1) = \min(\bar{x}(1), Y(1))$$

Since $x(1) \geq 2 \times \bar{x}(1)$, we have that $r(1) \geq \bar{r}(1)$, so $R(1) \geq \bar{R}(1)$, and the result holds.

Consider the following three cases. Since $x(1) \geq 2 \times \bar{x}(1)$, Y(1) can be in 3 different regions based on the relationship between $x(1)$ and $\bar{x}(1)$: $Y(1) \geq x(1) \geq \bar{x}(1)$, $x(1) \geq Y(1) \geq \bar{x}(1)$, and $x(1) \geq \bar{x}(1) \geq Y(1)$.

1. $Y(1) \geq x(1) \geq \bar{x}(1)$

   Since $Q_i(k) = Y_i(k) - x_i(k), \bar{Q}_i(k) = Y_i(k) - \bar{x}_i(k)$

   $$Q(1) \neq 0, \bar{Q}(1) \neq 0$$

2. $x(1) \geq Y(1) \geq \bar{x}(1)$

   $$Q(1) = 0, \bar{Q}(1) \neq 0$$

3. $x(1) \geq \bar{x}(1) \geq Y(1)$

   $$Q(1) = 0, \bar{Q}(1) = 0$$

Assume now that in the $(k-1)^{th}$ sub-period, we have $R(k-1) \geq \bar{R}(k-1)$. There are also three cases based on the queue length, similar to the $k = 1$ case.

1. $Q(k-1) \neq 0, \bar{Q}(k-1) \neq 0$
2. $Q(k-1) = 0, \bar{Q}(k-1) \neq 0$
3. $Q(k-1) = 0, \bar{Q}(k-1) = 0$

Consider what happens during the k^{th} subperiod, next.

1. Case $Q(k-1) \neq 0, \bar{Q}(k-1) \neq 0$. Since under CASTS algorithm the maximum number of requests it sends is the total number that left, so if the queue is not 0, it must have used up all credits available. Then

$$x(k) = 0, \bar{x}(k) = \lfloor \frac{X \times T}{K} \rfloor$$

$$r(k) = 0, \bar{r}(k) \leq \lfloor \frac{X \times T}{K} \rfloor$$

$$R(k) = \lfloor X \times T \rfloor, \bar{R}(k) \leq K \times \lfloor \frac{X \times T}{K} \rfloor \leq \lfloor X \times T \rfloor$$

   so $R(k) \geq \bar{R}(k)$, and the result holds.

2. Case $Q(k-1) = 0, \bar{Q}(k-1) \neq 0$. In this case, we can write

$$x(k) = \lfloor X \times T - R(k-1), \bar{x}(k) = \lfloor \frac{X \times T}{K} \rfloor$$

   $r(k) = \min(\lfloor X \times T - R(k-1) \rfloor, Y(k))$

   Now, if $\bar{Q}(k-1) \geq \bar{x}(k)$, we have

   $\bar{r}(k) = \bar{x}(k) \leq \bar{Q}(k-1)$

   because $R(k-1) + Q(k-1) = \bar{R}(k-1) + \bar{Q}(k-1)$

   $R(k-1) + 0 \geq \bar{R}(k-1) + \bar{r}(k) = \bar{R}(k)$

   so $R(k) \geq \bar{R}(k)$, and the result holds.

   If, on the other hand, $\bar{Q}(k-1) < \bar{x}(k)$, then we can write

   $\bar{r}(k) = \min(\lfloor \frac{X \times T}{K} \rfloor - \bar{Q}(k-1), Y(k)) + \bar{Q}(k-1)$

   Suppose that $\lfloor X \times T - R(k-1) \rfloor \geq \lfloor \frac{X \times T}{K} \rfloor - \bar{Q}(k-1)$. Since

$$R(k-1) + Q(k-1) = \bar{R}(k-1) + \bar{Q}(k-1) \; and \; Q(k-1) = 0$$

   we have

$$R(k) = R(k-1) + r(k) = R(k) + \min(\lfloor X \times T - R(k-1) \rfloor, Y(k))$$

$$\bar{R}(k) = \bar{R}(k-1) + \bar{r}(k) = \bar{R}(k-1) + \bar{Q}(k-1) + \bar{r}(k) - \bar{Q}(k-1)$$

$$= \bar{R}(k-1) + \bar{Q}(k-1) + \min(\lfloor \tfrac{X \times T}{K} \rfloor - \bar{Q}(k-1), Y(k))$$

Therefore, $R(k) \geq \bar{R}(k)$, and the result holds.

Suppose next that $\lfloor X \times T - R(k-1) \rfloor < \lfloor \tfrac{X \times T}{K} \rfloor - \bar{Q}(k-1)$.

We have to consider again two cases. First, if $Y(k) \geq \lfloor X \times T - R(k-1) \rfloor$, then

$R(k) = \lfloor X \times T \rfloor$, $\bar{R}(k) \leq K \times \lfloor \tfrac{X \times T}{K} \leq \lfloor X \times T \rfloor$

so $R(k) \geq \bar{R}(k)$, and the result holds.

Second, if $Y(k) < \lfloor X \times T - R(k-1) \rfloor$, then

$r(k) = Y(k)$, $\bar{r}(k) = Y(k) + Q(k-1)$

$R(k) = \bar{R}(k)$, and the result holds.

3. In case $Q(k-1) = 0, \bar{Q}(k-1) = 0$, we can write

   $x(k) = \lfloor X \times T - R(k-1) \rfloor$, $\bar{x}(k) = \lfloor \tfrac{X \times T}{K} \rfloor$

   $r(k) = \min(\lfloor X \times T - R(k-1) \rfloor, Y(k))$, $\bar{r}(k) = \min(\lfloor \tfrac{X \times T}{K} \rfloor, Y(k))$

   Now, if $\lfloor S \times T - R(k-1) \rfloor \geq \lfloor \tfrac{X \times T}{K} \rfloor$, then

   $r(k) \geq \bar{r}(k)$

   so $R(k) \geq \bar{R}(k)$, and the result holds.

   If $\lfloor S \times T - R(k-1) \rfloor \leq \lfloor \tfrac{X \times T}{K} \rfloor$, then

   if $Y(k) \geq \lfloor X \times T - R(k-1) \rfloor$, then

   $R(k) = \lfloor X \times T \rfloor$

   $\bar{R}(k) \leq K \times \lfloor \tfrac{X \times T}{K} \rfloor \leq \lfloor X \times T \rfloor$

   so $R(k) \geq \bar{R}(k)$, and the result holds.

   if $Y(k) < \lfloor X \times T - R(k-1) \rfloor$, then

   $r(k) = \bar{r}(k) = Y(k)$

   so $R(k) \geq \bar{R}(k)$, and the result holds.

In all cases above, the result holds.

A.3.4 The general case $B \neq 1, K \neq 1$

When $B = 1$, the result holds, i.e, it holds for each appliance. Treat every appliance independently, and substitute $x_i(k)$, $r_i(k)$, $Q_i(k)$, $R_i(k)$ for every $x(k)$, $r(k)$, $Q(k)$, $R(k)$ under both CASTS algorithm and the static algorithm in (3). We can get

$$R_i(k) \geq \bar{R}_i(k).$$

Since $R(k) = \sum_{i=1}^{B} R_i(k)$, and $\bar{R}(k) = \sum_{i=1}^{B} \bar{R}_i(k)$, we can get

$$R(k) \geq \bar{R}(k)$$

so the result holds.