MODEL-BASED DYNAMIC RESOURCE MANAGEMENT FOR SERVICE ORIENTED CLOUDS

HAMOUN GHANBARI

A DISSERTATION SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTORATE OF PHILOSOPHY

GRADUATE PROGRAM IN COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

SEPTEMBER 2014

Abstract

Cloud computing is a flexible platform for software as a service, as more and more applications are deployed on cloud. Major challenges in cloud include how to characterize the workload of the applications and how to manage the cloud resources efficiently by sharing them among many applications. The current state of the art considers a simplified model of the system, either ignoring the software components altogether or ignoring the relationship between individual software services. This thesis considers the following resource management problems for cloud-based service providers: (i) how to estimate the parameters of the current workload, (ii) how to meet Quality of Service (QoS) targets while minimizing infrastructure cost, (iii) how to allocate resources considering performance costs of virtual machine reconfigurations. To address the above problems, we propose a model-based feedback loop approach. The cloud infrastructure, the services, and the applications are modelled using Layered Queuing Models (LQM). These models are then optimized. Mathematical techniques are used to reduce the complexity of the models and address the

scalability issues. The main contributions of this thesis are: (i) Extended Kalman Filter (EKF) based techniques improved by dynamic clustering for scalable estimation of workload parameters, (ii) combination of adaptive empirical models (tuned during runtime) and stepwise optimizations for improving the overall allocation performance, (iii) dynamic service placement algorithms that consider the cost of virtual machine reconfiguration

First, to my family, whose support inspired my best efforts, and helped me through some challenging times. There are no words adequate for my appreciation. Second, to women in my larger family, in my motherland of Iran.

Acknowledgements

First, I offer my gratitude to my supervisor, Dr.Marin Litoiu whose support throughout my Ph.D. and his assistance especially during the final years of my Ph.D. studies has been vital. I attribute the level of my Ph.D. degree to his encouragement and effort; without him, this thesis not have been completed.

I warmly thank other members of Center of Excellence for Research in Adaptive Systems. Special thanks to my fellow Ph.D. student Cornel Barna for his constructive comments during my Ph.D., to Damon Sotoudeh for being my best schoolmate, and to Kent Poots for providing lots of positive feedbacks on the structure of the thesis.

Very special thanks to my greatest computer science teachers who truly made a difference in my life, Dr. Amir Amintabar, and Professor Mohammad Taghi Rouhani Rankouhi. I should also thank Prof. Stephen P. Boyd and Stanford University for providing several freely accessible online courses in the area of optimization, as well as their open-source software CVX.

I must also acknowledge Mohan Mishra, Sheila Wilmot and other York University graduate students union staff, for their critical assistance and help.

I owe my loving thanks to my parents, my mother Nahid and my father Behrooz, for supporting me throughout all my studies. I should also thank my mother and father-in-law Mahvash and Malek Niaz for their kindness and continued support.

Finally, and most importantly, I would like to thank my wife Elham for sharing six years of her precious life with me; for tolerating all of my shortcomings, and continuously injecting positive energy into our lives.

Table of Contents

List of Figures

List of Abbreviations

Acronyms

CBMG Consumer Behaviour Model Graph 32

EKF Extended Kalman Filter 21

HMM Hidden Markov Model 21

HPC high performance computing 26

HQN Hardware Queuing Network 36

IT Information Technology 14

JMX Java Management Extensions 16

LLC Limited Lookahead Control 24

LQG Linear-Quadratic-Gaussian 20

LSSC Large Scale Software Center 25

MAPE Monitor-Analyze-Plan-Execute 15

MDP Markov Decision Process 21

MOL Method of Layers 35

MVA Mean Value Analysis 28

POMDP Partially Observable Markov Decision Process 22

QN Queuing Network 35

SQN Software Queuing Network 35

ULS Ultra Large Scale systems 26

Math Symbols

$\Omega_{h,k}$ The multiplicity of resource type k of host h 108

λ Workload's average request arrival rate 29

$\lambda(\alpha_t)$ The cost of resources at time t. 116

y_{es} Mean requests made directly from any service e to service s 111

y_{es} Mean requests made directly from any service e to service s 32

σ_k The speed factor of resource k 34

R_c^Q The response time of class c requests considering the contention. 132

$R_{c,k}(\vec{N})$ Average delay for class c customer at center k for workload $\vec{N}$ 34

R_c^{SLA} SLO upper-bound on the response time of class c. 109

S The number of services in the data center. 108

T A lifetime of a cloud provider in the context of a control problem (possibly infinite). 115

t The timestep index 114

u_t^* Optimal placement action at time t 124

u_t Change in the placement made at time t. 116

$V_{c,s}$ The visit ratio of class c to service s. Or total direct and indirect mean requests to a service s for one request from a user class c 31

W_0 Workload component composed of intensities, think time, and demands for classes at time t 117

$X_c^Q(\theta^i)$ The throughput solution of the queuing model for a class c considering the contention when the placement is θ^i 132

$X_c(\vec{N})$ Average system throughput for class c customers for workload $\vec{N}$ 34

1 Introduction

1.1 Motivation and Approach

Cloud computing provides the computational power of data centers (e.g., the network, storage, computational devices, and services) to a large user community over the Internet.

A cloud is formed by an interconnected set of data centers. A data center is a set of physical machines (PM) interconnected by hierarchical network switches. The main operating costs associated with the data center are the cost of electricity and the cost of cooling. These costs are directly related to the number of active physical machines and switches (i.e., not in standby mode or powered-down).

Hardware virtualization multiplexes the hardware and offers small chunks of storage, CPU and memory in the form of Virtual Machines (VMs). In an Infrastructure as a Service (IaaS) cloud, the allocation of these virtual computing resources is controlled by centralized management software. The actual placement of VMs on PMs is usually hidden from upper layer entities.

Data center resources are used to provide cloud IT services[1]. These services can include everything from distributed file systems to business level software modules offered over the web. Services can be replicated to additional physical or virtual machines to increase the capacity for the incoming workload[2].

In the Platform as a Service (PaaS) form of the cloud, customers create and manage their own services. Customer services use generic services such as database, load-balancing, and messaging, offered by the PaaS provider. The decisions concerning deployment and resource allocation for services directly affect both the performance experienced by end-users and the cloud provider's cost of operations.

A major problem for Software as a Service (SaaS) and PaaS cloud providers is the optimal allocation of hardware resources (including CPUs, networks, etc.) to services[3]. The amount of resource needs is fulfilled by choosing the optimal deployment and resource share of the services on the available hardware. This problem is generally known as Optimal Service Placement (OSP)[107]. For OSP, the optimality is measured in terms of meeting the promised Quality of Service (QoS) mentioned in the Service Level Agreement (SLA)[4] while minimizing the

[1]Thus, they are sometimes referred to as Multi Service Information System (MSIS)[60].

[2]Replication means having separate copies of the same service on different hosts.

[3]For the PaaS environments, this allocation problem only arises when the entity which controls the software deployment and configuration settings is the cloud provider.

[4]Formally, a SLA is a contract, which defines the relationship between a service provider and its clients that fully specifies all obligations for both parties, the price to be paid for the service(s) offered and associated penalties should obligations be unmet. It can be quite complex and compre-

overall cost. SLAs are based on performance metrics of the applications or classes of customers[5]. Performance metrics usually represent the time behaviour of the services, such as the average throughput, the mean response time, and the total percentage of requests rejected or not handled within a certain time limit. In a PaaS and SaaS, the infrastructure cost is associated with the number of active hardware components contributing to the cost of the electricity and cooling.

This thesis addresses the following three issues:

(i) We propose a new approach for estimation of workloads of user classes and demands of services. An important issue in using the Bayesian approach is convergence. In order for the model to converge, the number of classes relative to the monitored performance metrics should be kept under a certain limit. We present an algorithm to reduce the number of classes by dynamically grouping them while keeping the number of classes high enough to achieve the accuracy objective. A combination of a clustering algorithm and filtering is proposed for effective grouping of classes of services. The grouping improves the earlier filter based approaches in terms of computational complexity and monitoring overhead.

(ii) We investigate if an empirical model, dynamically tuned through an Ex-

---

hensive (e.g., considering aspects of both functional and non-functional requirements); however, in this work, only performance objectives that can be extracted from a SLA are considered. No attempt is made to fully model or develop a SLA or a SLA management framework.

[5]A class of users in the context of this research is composed of a set of users that access the system services using the same pattern.

tended Kalman Filter, can outperform one that is obtained by an off-line regression analysis of applications' performances. Our contribution is to increase the allocation performance, by tuning the empirical models and achieving a better accuracy using the measurement data at runtime.

(iii) In the presence of reconfiguration costs, modeling the time (dimension) is inevitable. We propose a novel solution for long-term OSP, considering the reconfiguration cost. The solution uses the Model Predictive Control (MPC) framework (i.e. a branch of control theory). Details of the contributions of this thesis are presented in the sections which follow.

1.2 Contribution 1: Improving The Convergence of Performance Model Estimators using Dynamic Clustering of User Classes

Optimal allocation of computing resources typically requires some prior knowledge of the workload. The expected workload of user classes, and demands of services on infrastructure resources should be known beforehand. Usually these measurements are unknown because monitoring them would introduce lots of extra overhead. Thus, there is a need for an estimator that derives these parameters using the data obtained from the service center in an ongoing fashion and with minimum overhead.

Usually, the measured data includes the response times and throughputs of classes and per-server resource utilizations.

Least squares regression-based estimation (LSE) can be used to discover per-class resource demands. For example, having utilization and throughput, linear LSE [76] and having response time and throughput, nonlinear LSE [67, 84, 85, 105] have been used for estimation of service demands. The estimation process becomes more difficult if the collected data is incomplete (with respect to the model), and in reality, this is usually the case. For example, response time metrics and the number of invocations for backend services may be missing.

In the case of missing measurements, Bayes filters have been successfully applied before in [101, 102, 109]. In the Bayesian approach, hidden performance parameter values and missing data are both treated as unknown parameters to be estimated from observations. The probability estimates of the parameters and the missing data are updated as additional observations are made. In addition, the trade-off between trusting the old estimates and new measurements is addressed using control parameters of the filter.

An important issue in using the Bayesian approach is convergence. According to [108] a Kalman Filter[97][6] adopted for performance estimation does not converge to a solution unless the model has more measured parameters than estimated state

[6]A Kalman Filter is a Bayes filter with linear equations and normally distributed variables, and it is often used in control systems for estimation.

parameters[7]. As will be discussed in detail in Chapter 4, this puts a hard constraint on the number of classes that can be modelled. For each additional class, there needs to be some additional measurements from the system. For a large-scale system with many classes, measuring and storing an large amount of monitoring information at all times for the filter is impractical. This reduces the applicability of the Bayes Filter estimation for large service centers.

We propose the clustering of user classes into a smaller set with lower cardinality to reduce the measurement overhead and increase the scalability of the Bayesian performance model estimation. By decreasing the number of classes, we also reduce the estimator computation.

However, we cannot cluster all the classes into one group because then the model loses its detail and accuracy. We propose an algorithm to determine the best choice of the number of clusters and the grouping of classes into these clusters. The derived number of clusters guarantees that the monitoring overhead is reduced and at the same time, the model's accuracy is maintained. We use a concept called the *modeling error*, which is a heuristic measure to compare two different groupings of classes in terms of estimation accuracy[8]. We then propose an algorithm that reduces

[7][108] derives this as follows: the observability of the linear systems requires the matrix C in the observation equation of the system to have row rank of n, where n is the dimension of the state vector. Simply, C has to have n linearly independent rows, or LQM has to give n independent equations whose left hand side are monitored metrics.

[8]Modeling error is the mismatch of model output and the monitored metrics. It comes from the fact that the service demands are now represented using fewer parameterized statistical distribu-

the estimation complexity and overhead by grouping the classes, while keeping enough classes to maintain the modeling error below a prescribed threshold. The major contribution of our approach is this dynamic clustering of user classes, to reduce the estimation complexity yet leaving enough classes to satisfy the accuracy criterion.

1.3 Contribution 2: Optimal Resource Share Adjustment in Cloud Using Dynamically Tuned Empirical Models

A major research problem in the area of application service centers involves the allocation of limited resources to a set of applications, deployed on virtual machines (VMs). In a virtualized data center, resource allocation can be done by tuning the fraction of resource capacity allocated to each application's VM. Deriving the optimal set of resource fractions is usually done by optimizing a monolithic objective function based on the desired performance of applications. A model of the application service center is necessary for this optimization. The model can be either first principle (e.g. based on queuing theory) or be derived empirically from a data set.

A portion of the related work on optimal deployment such as those by Li et

tions. Note that this modeling error is a measure that we introduced. The Bayesian filter already tries to minimize the weighted cumulative sum of process noise and the observation noise over an infinite horizon of time. However, the modeling error that we define here is more a heuristic measure to compare two different groupings of classes over a very limited time horizon.

al.[58, 59] assumes that the system follows an accurate first principle model. Li et al. suggest using a filter-based approach such as [110] to estimate the parameters of this model adaptively. However, there is no guarantee that these first principle models accurately represent a service center since service center modeling is complex. Such modeling can consider the software contention, concurrency, remote calls, the limited amount of memory, and the limited amount of the network bandwidth.

On the other hand, there is a major problem with the empirical regression-based models. These empirical models do not typically use feedback of runtime performance data. When applications are deployed in a production environment, the models gradually become inaccurate. The inaccuracy is mainly because the deployment conditions are different from the test environments, used to build the training data sets.

Our contribution is to increase the accuracy of the empirical models by dynamic tuning using the measurement data at runtime. We also trace the accuracy increase in the overall performance of the allocation. We investigate if a model, built off-line through a nonlinear regression and dynamically tuned through an Extended Kalman Filter, can outperform a model that is not tuned. The other differences between our approach and the related work are the use of decomposed models[9], and a customized optimization routine to optimize the defined utility functions.

[9]In decomposed models, the overall cloud model is decomposed into individual models for applications.

We show that using static non-tuned application models results in suboptimal resource allocation for some applications, and leads to failure in meeting their SLAs. In contrast, dynamically tuned models result in more efficient resource allocations and better commitment to SLAs.

1.4 Contribution 3: Optimal Service Replica Placement via Model Predictive Control

Application optimization through service replication and allocation can yield frequent changes in deployments. The unnecessary frequent changes in the number of active servers or in the deployment of services on these servers can create overheads and perturbations for the running applications. Thus, if one of the objectives is to minimize the reconfiguration, or service replica movement, a typical stepwise optimization (i.e. an optimization that does not consider a larger time period and only targets one interval) does not work well.

To alleviate the excessive reconfiguration problem, we propose a dynamic service placement algorithm that considers the reconfiguration cost in calculating the optimal configuration using Model Predictive Control (MPC). The algorithm solves a finite-horizon deterministic control problem at each control step. The solution of the optimization problem includes the number of service replicas and optimal

deployment of replicas on available hardware at each step.

Through experiments, we validate our hypothesis that our model predictive approach performs better than the simple stepwise optimization (i.e., the second contribution noted) when the objective functions are long term and when reconfiguration costs are taken into account.

1.5 Dissertation Outline

The remainder of the thesis is structured as follows:

Chapter 2, Background, introduces the necessary background, including elements of the autonomic computing and adaptive systems, MAPE-K loop, optimal control theory, cloud computing, Layered Queuing Models (LQM), Bayesian estimation, and Kalman filter.

Chapter 3, Related Work, describes the state of the art for the problems of scalable performance parameter estimation, resource allocation using dynamically tuned application models, and service replica placement considering the reconfiguration cost. The related work for the first problem (i.e. service demand estimation), is classified into four categories: stepwise estimation, linear regression, nonlinear regression, and Bayesian estimation. We then provide a literature review of the second problem (i.e. optimal resource share adjustment in cloud using dynamically tuned empirical models). Third, we provide a literature review for the service

replica placement problem considering the reconfiguration cost. Since this is a new research topic, the related work is quite limited. We also provide a review of other papers in computing resource management area that used the MPC framework.

Chapter 4, Tracking Adaptive Performance Models using Dynamic Clustering of User Classes, introduces the first contribution, improving the scalability of the estimator while minimizing the monitoring overhead. Improving the estimator scalability is done by dynamic grouping of the classes of service. Then we prove the applicability of the approach through a set of tests which use standard industry benchmark data. We use the FIFA98 workload and the TPC-W benchmark for the experiments. We also perform a set of simulations to assess the result of the estimation and clustering algorithm for highly variable demands.

Chapter 5, Optimal Resource Share Adjustment in Cloud Using Dynamically Tuned Empirical Models, proposes the use of dynamically tuned empirical models within a resource share optimizer system. We first formulate the problem, introduce the modeling and estimation, and then introduce the optimization process. We claim that the dynamically tuning empirical performance models of the applications, despite their computational complexity, can improve the quality of the allocation. This claim will be assessed by two cases studies of different scales.

In chapter 6, Optimal Service Replica Placement via Model Predictive Control, we target the optimal service placement problem considering the reconfiguration

cost. We propose a solution to the problem via the Model Predictive Control framework. There is a set of challenges in solving the introduced problem, including the non-linearity of the LQM and the non-linearity of the infrastructure cost. We provide an efficient solution to the problem. The solution is transformed into a solver friendly format that can be handled by a common convex optimization solvers such as `cvx`. We examine the behaviour of the resulting optimization in a set of simulated experiments using the FIFA98 workload and a synthetic service center. In the a second set of simulations, we investigate the effect of the lookahead horizon in the performance of the allocation. The results are presented as a set of cost trade-off curves that compare the variation of the cost factors (i.e. the cost of SLA violations, the cost of resource, and the cost of reconfiguration) for different lookahead horizons.

Chapter 7, Summary and Conclusion, concludes and discusses possible future work.

2 Background

This chapter introduces the necessary theoretical background for the rest of the dissertation. First, we introduce autonomic computing and adaptive systems. These are explained in the context of the MAPE-K loop, a guideline for designing autonomic systems. We also introduce some necessary components from control theory. We review optimal linearized control, policy iteration, sub-optimal heuristic control, steady-state equivalent problems and model predictive control (MPC) as different ways to construct control loops. Finally, we introduce Layered Queuing Models (LQM) as a tool to model the performance of clouds. We also provide an explanation about the way we map the cloud concepts (i.e. end-users, services and invocations, infrastructure, and replication) to the model elements (i.e. user classes, delay and queuing centers, and service demands, etc.). We then review the solution techniques for the closed queuing networks considering software contention and multiple resources.

2.1 Elements of Autonomic Computing and Adaptive Systems

Autonomic computing, a term introduced by IBM [50], relates to systems that are self-managing, self-tuning, self-healing, self-protecting, self-adapting, self-configuring, and self-organizing (briefly called self-* systems) [16]. Some examples of IT related self-* areas of research are: adaptive parameter-level configuration management [27, 28, 33], adaptive client-server communication [20, 66, 75], adaptive resource allocation [31, 56], self-configuring network services [43], workload adaptive services [22], self-managing storage [73], statistical inference based decision-making management [29], and change and configuration management [96].

The autonomic computing subspecialty relevant to this work is the building of self-optimizing complex resource sharing systems that manage themselves in accordance with specific high-level objectives [51]. The autonomic aspect focuses on the fact that the management is performed by the systems themselves rather than being actively performed by human operators. This management is to be in accordance with system management objective set by administrators. An autonomic manager typically understands the desired system objectives, matches those objectives with the current or forecasted system behaviour, and incorporates the objectives into its decisions governing the system.

There are options in considering autonomic management system architecture. In the architecture considered in this research, adaptation strategies and mechanisms are separated from the applications or systems [37, 92]. The architecture used is the well-known Monitor-Analyze-Plan-Execute (MAPE) loop suggested by IBM [50], where several components such as an analyzer, automated learner, forecaster, and a planner are used to decide proper actuator action(s) given the current system measures. Figure 2.1, adopted from [2], presents a schematic structure of such loop.

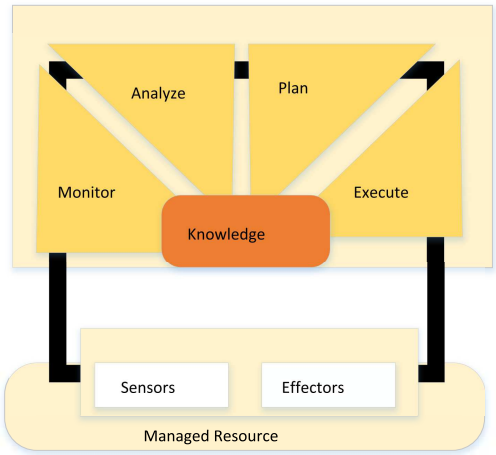


Figure 2.1: Architecture of autonomic management loop suggested by IBM (adopted from [2]).

Components of the autonomic loop have a specific interpretation when viewed in the control theory context. The following is an enumeration (description) of autonomic loop components considering control theory. In a control theory sense,

the goal of any optimal control loop is to choose a sequence of feasible *control actions* over time that maximize a defined *performance criterion* (or objective function)[10].

2.1.1 Monitoring Subsystem

The monitoring subsystem is responsible for measuring inputs, and outputs of the managed system (system I/O in a control theory sense), quantifying system I/O, sometimes aggregating system I/O, and keeping system I/O as a history. In a closed-loop (or feedback based) control, the controller constantly considers the most recent monitored data for calculating the proper actions.

Several computer system performance metrics can be collected. For example, hardware level metrics for each host, operating system metrics (e.g. file, network, and memory management subsystems), process level metrics, and service specific metrics for specific types of server processes (e.g. load balancer, web server, application server, and the database server). In a batch-oriented system there are additional metrics, such as the average number of jobs in a queue or the average job's queuing time. There are existing tools for monitoring computer systems, such as *Collectd* [3], *Nagios* [7], Java Management Extensions (JMX) [6], and IBM Tivoli monitoring[5].

[10]One can instead say the optimal control minimizes an expected total cost function.

2.1.2 Analyzer

The analyzer subsystem of the autonomic management loop, identifies and tunes a model of the system, and estimates the unobservable portion of the system state. The model enables the autonomic manager to project the system's behaviour and the state, from the current state under different actions. A major concern is the ability to synchronize the model based on observed behaviour of the system. In our case, the analyzer is composed of a first principle mathematical model encoding the system's knowledge combined with statistical techniques to perform data-driven learning.

2.1.3 Planner

A planner uses the model to rapidly explore multiple decisions and find near-optimal solution [13, 62]. The search space is formed by the actions over time. The search for finding the proper actions can be based on a mathematical optimization in continuous space (e.g. methods such as primal decomposition, interior point methods, stimulated annealing, etc.) or can be a search in the discrete space (e.g. a combinatorial optimization). Whether the search is continuous or discrete, it is directed towards a goal or a set of goals.

2.2 Optimal Control Theory

In general, the goal of any optimal control approach is to choose the sequence of feasible control actions that maximizes a defined performance criterion (or an objective function) of the following form:

$$\underset{u_0,\ldots,u_{T-1}}{\text{minimize}} \lim_{T\to\infty} \frac{1}{T} E\left[\sum_{t=0}^{T-1} g(x_t, u_t)\right] \tag{2.1}$$

$$x_{t+1} = f(x_t, u_t) + w_t \tag{2.2}$$

$$h(x_t, u_t) < 0 \tag{2.3}$$

where x_t is the system state, $E[\ldots]$ denotes the expected value, u_t is the controlled input, w_t is the uncontrolled stochastic input, T is the control interval, g is the (non)linear stage cost function, f is the (non)linear state transition function which maps the current state to the next one, h is the nonlinear function defining the limits of possible inputs and states. Equation 2.1 denotes that the objective is to minimize the overall cost, from the current time step to infinity (i.e. infinite horizon control), by choosing the optimal control inputs. The equation 2.2 is called the process model and determines how the system state moves from one step to the next. The inequality 2.3 defines a set of possible state-input combinations at any given time. For example, if at state x_1 input u_1 is allowed, then (u_1, x_1) will be in the set representing the h.

In a feedback based scheme or a **closed loop control**, at any given time, the

controller makes its decisions (i.e., to allocate resources) based on the information available from the system up to that time. The solution to the above optimization is obtained and applied at each step of the control. Thus, the original plan is adjusted according to the new observation samples of the environment from the previous step. This solution is called a control policy:

$$\phi_t = \mathbf{R^{(t+1)n}} \rightarrow \mathbf{R^m}$$

$$u_t = \phi_t(x_0, ..., x_t)$$

where n is the number of the state variables, m is the number of the input variables, and t is the index of the current time step.

Normally for solving this optimal control problem one must form a set of equations according to dynamic programming principles. For most cases, the dynamic programming is often quite challenging and impossible for large problems. In the following subsections, we briefly summarize those control problems of interest that can be solved analytically or computed numerically with polynomial complexity. These control problems include linearized dynamic control, optimal linearized dynamic control, policy iteration in finite state models, heuristic policies, and model predictive control (MPC). The proposed controllers vary in their architectural complexity; which depends on the existence of a model for the target system, the theory underpinning the model, and the usage (or omission) of an estimator to track hidden

variables in the model.

2.2.1 Optimal Linearized Control

In some control problems, the optimal control policy has static state feedback control form: $u_t = \psi_t(x_t)$. Using this policy, an optimal input at every step can be obtained from the current state (as opposed to the current and previous states $u_t = \phi_t(x_0, ..., x_t)$). For example, consider the Linear-Quadratic-Gaussian (LQG) control problem:

$$\underset{u_0,\ldots,u_{T-1}}{\text{minimize}} \lim_{T\to\infty} \frac{1}{T} E\left[\sum_{t=0}^{T-1} l(x_t, u_t)\right] \tag{2.4}$$

$$x_{t+1} = Ax_t + Bu_t + w_t \tag{2.5}$$

$$F_x x_t + F_u u_t < f \tag{2.6}$$

Here the symbols x_t, u_t, w_t, T have the same meaning as equations 2.1 to 2.3; they subsequently denote the system state, control input, stochastic input, and system lifetime. l denotes the Euclidean (or second) norm function (i.e. $\|\boldsymbol{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}$). F_x and F_u are convex functions of x and u. f is a convex function. Finally, A and B are matrices.

The equation 2.4 denotes that the objective is to minimize the overall cost from the current time step to infinity (i.e. infinite horizon control), by choosing the optimal control inputs. The only difference with 2.1 is that here the stage cost

function l is quadratic. Note that, the process model denoted by the equation 2.5 has a linear form. Also note that, the set of possible state-input combinations, defined by the inequality constraint 2.6, is guaranteed to be convex.

The solution (i.e. the control signal) for this problem, has a feedback form, and is itself a linear system called Linear Quadratic Regulator[4]. It takes the output of the system under control (y_t) as input and generates the proper input to be applied to the system (u_t). Internally, this controller is based on a dynamically adjusted LQ-optimal gain applied on the *state estimate* driven from an estimator. The estimator in LQG is usually a simple or an extended Kalman filter [98] able to maintain a good estimate of model unknowns by calibrating itself to measurements during runtime. Note that the LQG solution is only optimal for the linear dynamic systems with Gaussian noise, quadratic objectives and no constraints.

2.2.2 Policy Iteration

When the model underlying the system is Markovian with finite states X, the problem of finding an optimal policy that maximizes a long-term cost function, has a solution; investigated under the umbrella of Markov Decision Processes (MDP). Computing the optimal value function and policy requires solving a nonlinear system of equations. When the underlying state of the system is hidden (Hidden Markov Model or HMM), which is usually the case, solving the problem is not

quite so simple. The solution can then be investigated as a Partially Observable Markov Decision Process (POMDP). Unlike LQG, in a POMDP, there is no separation principle. Meaning that, one cannot estimate the state distribution and perform optimal control on the known state.[11] The solution, in this case, is to pick a policy format and gradually improve the stationary policy using the observed result of the controller. The *policy iteration algorithm* generates a sequence of improving stationary policies. Similarly, the *value iteration algorithm* is then used to converge iteratively towards an optimal cost J^*.

2.2.3 Sub-optimal Control

There is a small set of problems for which the optimal control policy with a feedback control form can be computed. However, sub-optimal feedback solutions are extremely common. For example, consider a linear system which is mathematically modelled as:

$$x_{t+1} = Ax_t + Bu_t$$

where $u_{t'}$ are the control inputs and $x_{t'}$ is the system state. In the *linearized dynamic control*, one should suggest a small or efficient set of control inputs u that transfers x to x_{target}. Similarly, a regulation problem involves taking the system

[11] In LQG the state distribution is Gaussian, and thus, the mean value can be easily used for projecting to future.

output (y_t) to a reference output or a set-point (y_{target}). Assuming the control error is the difference between the current system output and the desired output (y_{target}): $e_t = y_t - y_{\text{target}}$, it is desired that control error be transferred to zero.

A feedback loop for this purpose can be constructed by feeding back the control error (the difference between a set-point and a measured output) as an input to the system (often with some intermediate processing). For a system whose dynamics are known [12](e.g. through system identification techniques), a proper feedback based regulator can be designed to make the system output achieve a degree of the following properties: (i) responsiveness to an error, (ii) the degree to which it overshoots the set-point, (iii) the degree of oscillation.

For a system whose dynamics are known (e.g. through system identification techniques), a proper controller can be designed using the design techniques such as pole placement, root locus, etc. The model used during system identification can take several forms such as transfer functions, state-space models, difference equations, etc.

[12]The dynamics of a system can be represented in several mathematical forms such as transfer functions, state-space models, difference equations, etc. Depending on the form used, the controller might be designed using the same mathematical method (e.g. a transfer function).

2.2.4 Model Predictive Control (MPC)

In Model Predictive Control (MPC) (also referred to as Limited Lookahead Control or LLC)[11, 48], the controller explores a search space formed by different choices of control actions over a predicted model [23] to find an optimal solution; thus, a management problem is posed as a sequential optimization under uncertainty. The search space includes a set of future states within a lookahead horizon. Controller then selects a path that minimizes a cumulative cost while satisfying both state and input constraints within the lookahead horizon $u_j^*|j \in [t, t+N-1]$.

In a Certainly Equivalent Controller (CEC) implementation, at each time step t, one makes a set of forecasts $\hat{w}_{t|t}, \ldots, \hat{w}_{t+N-1|t}$ for future steps based on the observed state trajectory $x_0, ..., x_t$. Then one is simply going to use these forecasts in the following optimal control problem as if the forecast were perfect. The first control input leading to this path is chosen as the next control action.

$$\underset{u_t,\ldots,u_{N+t-1}}{\text{minimize}} \sum_{j=t}^{t+N-1} l_t(x_j, u_j) + l_{t+N}(x_T) \tag{2.7}$$

$$\text{subject to: } x_{j+1} = Ax_j + Bu_j + \hat{w}_{j|t} \qquad j = t...t+N-1 \tag{2.8}$$

$$u_j \in U_j \qquad j = t...t+N-1 \tag{2.9}$$

Here $u_t, \ldots, u_{N+t-1}$ are variables and $x_t, \hat{w}_{t|t}, \ldots, \hat{w}_{t+N-1|t}$ is the known data. The symbol x_t has the same meaning as equations 2.4 to 2.6; it denotes system state at time t. N is the length of the lookahead window, $\hat{w}_{j|t}$ is the prediction of future

disturbance w_j at time j. u_j is the future control input at time step j. $l_t(x_j, u_j)$ is a quadratic function representing the stage cost at time step t. $l_{t+N}(x_T)$ is the cost of deviation of the terminating state x_T. U_j is the input constraint at future time j.

There is a common misconception about model predictive control, that the quality of the derived control inputs only depends on the quality of the predictions. However, this is not true. In MPC, one forms a prediction of what the future disturbances $\hat{w}_{t|t}, \ldots, \hat{w}_{t+N-1|t}$ are going to be, based on the observed state trajectory $x_0, ..., x_t$. These predictions can be just the mean value of the disturbance (i.e. as opposed to conditional expectation), or they can come from an external analyst. No assumption regarding the correctness of the forecasts is necessary for CEC to work. In most cases, not one of the $\hat{w}_{j|t}$ will actually come true; despite this, the approach works very well in practice. In fact, if all $\hat{w}_{j|t}$ were true, then the problem would be a deterministic optimal control (as opposed to stochastic control problem).

2.3 Cloud Computing

A Large Scale Software Center (LSSC) aims at large scale delivery of on-demand computational power (specifically network storage, computational devices, and ser-

vices) to a user community. [13] Current examples of such system include grid computing [34, 35] and cloud computing [15, 25, 42, 83].

Grid computing, having arisen from the high-performance computing (HPC) community near 2000, targets the delivery of on-demand computational power using a unified network of loosely coupled computers in the form of "super virtual computer". The aim of designers was to let users plug their own programs into the infrastructure, and use resources for a desired duration. Current implementations of computing grids usually target long-running resource intensive applications (or jobs) that are submitted by a few users. Examples of these applications are distributed simulation [93, 94], scientific visualization [100], continual queries [17, 53], video conferencing [44], and transcoding [8].

Cloud computing is another manifestation of LSSC that, according to many, has caught on in mainstream enterprises. In the cloud, there is usually a clear distinction between a provider and a customer, and the extent of sharing is governed by economic rules and pricing (usually pay-as-you-go). In cloud computing, large condensed data centers, possibly operated by multiple stakeholders, are offered at different levels of abstraction (e.g. infrastructure (IaaS), platform (PaaS) and software (SaaS)) on-demand as commodities to a large community of users. While

[13] Ultra Large Scale systems (ULS), is a closely related term coined by researchers at Carnegie Mellon's Software Engineering Institute, referring to a system composed of a large set of systems with a variety of stakeholders communicating and operating to satisfy separate (possibly conflicting) goals.

IaaS focuses on offering virtualized hardware, PaaS hosts applications composed of services, which make use of other infrastructure services. In PaaS, usually a specific programming language and API for infrastructure services is enforced. In SaaS, there is a more customized set of APIs for specific types of applications (for example, accounting), shared by all applications of that type.

The cloud model we consider in this thesis is composed of a set of applications, offering services that are used by end-users. The end-users are usually service consumers around the globe accessing the services through Internet Protocol (IP), and most often Hypertext Transfer Protocol (HTTP). End-users, in our cloud model, are divided into a set of classes, where each class has a certain population and behaviour (in terms of accessing the cloud resources). The services are either generic and offered by the cloud provider (such as naming, storage, etc.) or application-specific and developed by cloud consumers. Both services are hosted and administered on the available infrastructure. Some services developed by the consumers are frontend, meaning that they are the first point of interaction with the end-user, other services are backend, meaning that they are invoked by frontend services.

In our cloud model, the relationship between classes and applications is one-to-one: (i) each class uses the limited set of services offered by a single application (ii) there is only one class of users associated with each application. Thus, in this thesis, we can use the concepts of an application and class of users interchangeably.

2.4 Service Centers as a Layered Queuing Models (LQM)

Queuing theory based models [18, 78, 80] and Layered Queuing Models (LQM) [81, 86] developed upon Mean Value Analysis (MVA) of queuing networks have been used to capture the behaviour of multi-tier distributed applications [38, 62, 64, 103]. Queuing Models can be utilized to describe the expected performance of service centers (i.e. response time and throughput) in relation to various inputs. In the following subsections, we review the components of the queuing models.

2.4.1 Workload

In this section, we review workload parameters derived from a workload characterization, including workload intensities and service demands.

2.4.1.1 Workload Intensities

Software components hosted in the cloud can be categorized Workload of a service is driven by its external users but workload of a standalone software component (such as a simulator or a batch data processor) is controlled by itself or its data. Both types of software components can be generalized as one. In the first case, the workload is specified as a *number of users* or a *request rate*, and in the second case it is usually specified as a *multiprogramming level* (the number of simultaneous

threads running on behalf of a process) or an *execution frequency* (the frequency that a specific software module is called). The number of users for a service is equivalent to the multi-programming level and the user request rate is equivalent to the execution frequency.

In general, the workload component can be modelled as either *closed* or *open*. In the open model, customers are assumed to make requests on average every τ seconds, independent of when they receive responses for the previous requests. The open workloads are identified by an average request inter-arrival time (τ), which is measured in seconds, or an arrival rate (λ), which is measured in requests per seconds. Often, these models assume that the arrival rate (λ) is a homogeneous Poisson process, and that inter-arrival times (τ) are exponentially distributed with parameter λ (mean of $1/\lambda$). In a closed model, clients are assumed to wait for responses to their requests. Upon receipt of this response, the client spends some time determining the next action before issuing a further request. Closed models are defined by the number of users (N) and average think time (Z). The think time is considered to have an exponential distribution.

A service center may offer a large number of services where each service is used by more than one group of users. Each group of users may consume the services and the resources in a different way (for example, with highly different CPU demands). Thus, it is common to refer to each group as a different **class**. In multi-class models,

outputs are given in terms of the individual customer classes. It is, therefore, reasonable to model each application with a separate class of users. In the case of open multi-class models (with say C customer classes), workload intensity is denoted by $\lambda \equiv (\lambda_1 \dots \lambda_C)$, where λ_c, is a class c arrival rate. A closed multi-class model consists of C classes, each of which has a fixed population. We denote the workload intensity by $N \equiv (N_1 \dots N_C)$ and by its think time $Z \equiv (Z_1 \dots Z_c)$, where N_c is the class c population size, and Z_c is the class c think time.

2.4.1.2 Workload demand: Interaction with Resources

A service and an application can be modelled by a set of queues, where devices are mapped to queuing or delay centers. CPU is the main resource consumed by programs. Other resources such as hard disks and network are used from CPU. The programs accessing system resources are modelled as the customers of the queuing network. By modeling the cloud workload as a queuing network model, the general Mean Value Analysis (MVA) can be used to calculate the performance characteristics of the system.

In queuing models, service interaction with hardware resources is quantified in terms of service times. The service time $D_{s,k}$ is the time each request spends on a resource type k when accessing service s. For example, `view cart` service that is executed on an application server could be characterized by the following

parameters: $D_{\texttt{viewcart},\text{CPU}} = 30$ ms, $D_{\texttt{viewcart},\text{DISK}} = 30$ ms. Here, we assume the basic operations performed at the device when executing the service by the various classes are the same; thus, it is reasonable to assume that the average service times across classes are nearly equal. However, it is possible for different customer classes to require different total numbers of visits to the service center ($V_{c,s}$), thus providing distinct service demands (which we denote by $d_{c,s,k}$):

$$d_{c,s,k} = V_{c,s} D_{s,k} \tag{2.10}$$

here $V_{c,s}$ is the *visit ratio* of class c to service s. It represents the total direct and indirect mean requests to a service s for one request from a user class c[14]. $d_{c,s,k}$ is the total service demand the service s makes on a resource k by a single request of class c.

2.4.2 Deriving Visit Ratios

For some service-oriented systems, there is a systematic way to calculate $V_{c,s}$ as follows: $V_{c,s}$ for front-end services can be calculated by Consumer Behaviour Model

[14]If $V_{c,s}$ is normalized (i.e. $\sum_s V_{c,s} = 1$) then accesses to the S services fall into a distribution referred to as the *Web Interaction Mix*. For example, TPCW benchmark specifies two web interaction mixes: "One is intended to simulate a workload where there are few buy orders and the majority of the customer requests are browsing the website. This is accomplished by having 95% of the web pages accessed be the browsing pages, (Home, New Products, Best Sellers, Product Detail and Search pages) while only 5% of the Web accesses are to the order web pages. This mix tends to place more pressure on the front-end Web Servers, Image Servers and Web Caches. The second mix is intended to simulate a website with a significant percentage of order requests. This is accomplished by having 50% of the web page accesses be the browsing pages and 50% of the accesses be to the order web pages. The second mix stresses the Database Server."[10]

Graph (CBMG)[15] [70]. For backend services $V_{c,s}$ are obtained using the visit ratio of front-end services, and *call multiplicities* on the edges of *service call graph*. A service call graph is an acyclic directed graph where each edge represents the number of calls from a caller service to a callee and each edge is annotated by a call multiplicity. Using the call graph one can calculate the visit ratios. Let y_{es} be the mean requests made directly (i.e. call multiplicity) from service e to service s. Formally, the *visit ratios* satisfy the following equation:

$$\begin{cases} V_{c,s} = \sum_{s'=1}^{S} V_{c,s'} y_{s',s} & \text{where } s \text{ is a backend service} \\ V_{c,s''} \quad \text{is computed from CBMG,} & \text{where } s'' \text{ is a front end service} \end{cases}$$

Algorithmically, $V_{c,s}$ is calculated from the call graph y_{es} using the following set of equations[16]:

$$\begin{aligned} & con^0 = \mathbf{0}_{S+C,S+C} \\ & con^{(i)} = con^{(i-1)} + y^i \text{ for } i \in 1 \dots \text{levels} \\ & V = con^{\text{levels}}_{1 \dots C, C+1 \dots C+S} \end{aligned} \tag{2.11}$$

where $con^{(i)}$ represents the *contact level* of services and classes at the i'th iteration, taking y as the adjacency matrix and y^i is the power of y, or in simple words it is the contact level of each node to its i'th further away node. For example, if web

[15] CBMG is presented as a state transition graph in the workload characterization process.

[16] Assuming there are no request cycles, Y_{cs}.

service B and C provide services to A, and services B and C both use web service E, then the con^2 from A to E is calculated as $y_{AB}y_{BE} + y_{AC}y_{CE}$.

2.4.3 Replication

When services are replicated, multiple instances are deployed on multiple hosts. Each replica is indexed by $j = (s, h)$ where s represents the service and h represents the host for the replica. Here the load of the original service is distributed between the replicas. One can adjust the load on each replica by distributing the accesses to replicas in a desired proportion. This distribution can be modelled by changing the call multiplicities in the call graph in the edges between the callers and the callee service replicas, or by directly considering the call multiplicities in visit ratios $V_{c,s}$. Note that in case of the replication, service demands between replicas are shifted based on V as the equation 2.10 can be written as:

$$d_{c,s,k,h} = V_{c,s,h} D_{s,k}$$

$$V_{c,s} = \sum_{h} V_{c,s,h}$$

Also according to [72] the average service demand for a class c on a resource k of a host h can be obtained from the original service demands as follows:

$$d_{c,k,h} = \sum_{s} d_{c,s,k,h}$$

Here, it is assumed that service demands are independent of the class of service and the host where the service is deployed. Essentially, it is separating the servers and the resources such as CPU.

2.4.4 Solution Techniques for the Closed Queuing Networks

The set of equations giving the mean value analysis (MVA) solution to the model for closed workloads is as follows:

$$X_c(\vec{N}) = N_c(\vec{N})/(Z_c + \sum_{k=1}^{K} R_{c,k}(\vec{N})) \tag{2.12}$$

$$Q_{c,k}(\vec{N}) = X_c(\vec{N}) R_{c,k}(\vec{N}) \tag{2.13}$$

$$R_{c,k}(\vec{N}) = \frac{D_{c,k}}{\sigma_k}(1 + Q_k(\overrightarrow{N - 1_c})) \tag{2.14}$$

$$Q_k(\vec{N}) = \sum_{c=1}^{C} Q_{c,k}(\vec{N}) \tag{2.15}$$

Here, $X_c(\vec{N})$ denotes the average system throughput for the class c customers for the workload $\vec{N}$. $R_{c,k}(\vec{N})$ denotes the average delay for the class c customer at the center k for workload $\vec{N}$. $Q_{c,k}(\vec{N})$ denotes average number of requests of the class c in the queue of the resource k for the workload $\vec{N}$. $Q_k(\overrightarrow{N - 1_c})$ is the total queue length of the resource k for the predecessor workloads $\overrightarrow{N - 1_c}$ with one class c customer less than $\vec{N}$. σ_k denotes the *speed factor* of the resource k. Here, the speed factor is used to standardize different resources in terms of the speed. This is to make the demands machine and application independent. Demand is described

in terms of number of compute seconds for a standard CPU. Transferring a class's workload to different resources is assumed to preserve the standardized resource demand, although the final demand on the resource (i.e. $\frac{D_{c,k}}{\sigma_k}$) would depend on the resource's speed factor.

Efficient solutions can be obtained for the resulting set of equations by sequentially solving each equation for each workload $Q_{c,k}(N)$ based on predecessor workloads $Q_{c,k}(N - 1_c)$); as opposed to simultaneously performing Newton step on equations written for all workload components. Another approximate fast solution is to substitute the following equation for equation 2.14.

$$R_k(N) = D_k \left(1 + \left[\frac{N_c - 1}{N_c} Q_k(N)\right]\right) \tag{2.16}$$

One needs to initialize each $Q_k(N) = N/K$ for all k and iterate through the formulas until the Q_k's converge to a solution[17].

2.4.5 Software Contention and Layered Queuing Networks

A method for considering software contention is to use the extensions such as the Layered Queuing Model (LQM) and the Method of Layers (MOL) [86]. In these models, software and hardware resources are mapped to separate networks of queues. The software queuing network [86] (SQN) is constructed from the software

[17] Agrees to the last iteration within some tolerance (e.g., 0.1%).

call graph and hardware is derived from the computer architecture. The way SQNs are derived from call graph is as follows:

1. For the software service that is not a critical section or is not governed by a thread pool there will be a delay centre in the SQN whose delay depends on the resources the service uses from hardware queuing network (HQN).

2. For every critical section code (i.e. governed by a semaphore or a monitor) there is going to be a single queuing centre in the SQN.

3. For every software resource governed by a thread pool there is going to be a multi-server in the SQN.

4. For every call between a software service a and software service b, there will be a connection between the corresponding centres in SQN. Moreover, a call multiplicity is associated with every call.

The number of visits for every class and hardware-software resource is obtained by accumulating the call multiplicities as described earlier. The initial hardware-software service demands for each class is obtained by multiplying visit ratios and service times. The actual algorithm providing the solution to LQN is provided in chapter 8 (appendix 1).

2.4.6 Multiple Resources with a Shared Queue

Hardware and software resources with multiplicity (such as multi-core CPUs or even multiple identical CPUs with the same workload) are modelled with multi-servers. Multi-servers can be approximated with techniques introduced in [55, 71, 87] for load-dependent servers. Multiplicities for software resource are used to model multi-threaded software where active threads of the thread pool are modelled as multi-resources, sharing the same request queue. Note that a single active thread case (i.e. a critical section implemented by a monitor or semaphore) is modelled by a single software queuing center with multiplicity of 1. Also, note that the solution of the LQM with unlimited software resource multiplicity (i.e. thread pool-less or pass-through software modules) and unlimited hardware resource multiplicity (i.e. delay center) denoted by $RT_{c,\infty}$, is going to be sum of the demands of software services invoked by each request of the class c on the hardware resources:

$$R_c^{\infty} = \sum_k R_{c,k}^{\infty} = \sum_k D_{c,k} \tag{2.17}$$

2.4.7 Model Granularity and Modeling Complexity

In this thesis we mainly associate queuing and delay centers in our models with computing resources of servers (i.e. CPUs). However, during the estimation process, the service demands capture a lot of details other than just the CPUs processing

rate or delay.

In a typical data center, there are far more elements than just server CPUs. Some of these elements are physical machine's RAM and hard drives, storage area networks, communication network switches (i.e. top of the rack and aggregate switches), and server caches.

Although it is possible to model a large subset of these entities using queuing network models, this might not be useful in the later stages when one tries to estimate the models unknowns or optimize over controllable parameters of the model.

When each component of the data center is modeled using a separate queuing center, the estimation of parameters of such model becomes extremely hard. This is due to the fact that not all of data centers components provide measurements that can be used in the estimation process. For example physical level network links (i.e. wires), and data link layer Ethernet switches might not provide resource utilization information. This means that, we would not be able to know what percentage of the capacity of a physical wire is used or unused, and thus we have to incorporate this in the model as unknown parameters.

In our approach, each queuing center not only models the CPUs, but also abstracts a set of physical resources that the workload passes through while executing a specific service. For example, the delay of a queuing center that models a Web

server, already includes the delay added by the physical machines hard disks, remote storage, or virtual memory management unit (which itself is affected by the amount of RAM). In conclusion, the demands of this queuing centers capture a lot of details that might not be limited to host CPUs.

2.5 Summary

In this chapter, we reviewed the necessary theoretical background. We introduced the autonomic computing and the IBM's MAPE-K loop guideline. We also reviewed the mathematics behind several types of controllers. Finally, we described the mathematics of performance models for modelling software and hardware components. We explained how, different applications are modelled using customer classes, and how services are modelled using queuing and delay centres.

3 Related Work

The chapter reviews the current state of the art for solutions to the IT service resource management problems introduced in chapter 1. The problem solutions reviewed include: (i) methods for reducing monitoring overhead and computational complexity of the Bayesian service demand estimation for service replicas, (ii) methods for improving the placement of service replicas in a private cloud using dynamic empirical models, and (iii) methods for considering the reconfiguration cost, when modeling the optimal placement of the service replicas.

Separate sections are provided which describe the related work for each (i-iii) of these problems.

3.1 Service Demand Estimation

Some early research into CPU demand estimation is described in [67]. In this early work, missing service demands are computed by solving a set of nonlinear equations

based on a queuing model. For each step, the following equations are solved:

$$\underset{(d_{c,k})}{\text{minimize}} \; ||R_c - \sum_{k=1}^{K} \frac{d_{c,k}}{1 - \sum_{c'=1}^{C} \lambda_{c'} d_{c',k}}||$$

subject to: (3.1)

$$d_{c,k} > 0 \text{ for each } c,k$$

$$\sum_{c=1}^{C} \lambda_c d_{c,k} < 1 \text{ for each } k$$

where R_c and λ_c are known and $d_{c,k}$ are estimated. To solve 3.1, Menascé et al. use an iterative approximation algorithm that takes the estimated demands of the past step as the initial value for the estimates of the current step.

[84] and [85] used linear regression to predict the demands of running programs in a distributed system. Many approaches, such as [76], estimate service demands by solving the following linear Least Square Estimation (LSE) problem based on the utilization law

$$\underset{(d_{c,k})}{\text{minimize}} \; ||U_{t,k} - \sum_{c} X_{c,t} d_{c,k}|| \tag{3.2}$$

here $U_{t,k}$ denotes the utilization of the k'th resource centre at time t, X_c denotes the throughput of the customer class c, $d_{c,k}$ denotes the class c's service demand on the k'th resource centre, and $||.||$ represents the second norm function. In the formulation 3.2, it is assumed that the throughput values of all user classes and the utilizations of all servers are known over some time interval. It is also assumed

that, the LSE is based on the utilization law and the assumption that residuals are independent and identically distributed (IID) noise with a Gaussian density (i.e. $N(0, \sigma^2)$):

$$U_{t,k} = \sum_c X_{c,t} d_{c,k} + v_{t,k},\ t = 1, ..., T, k = 1, ..., K \tag{3.3}$$

where T is the length of a monitored time interval (or the number of samples), K is the number of resources and $v_{t,k}$ is the residual associated with time t and resource k.

References [76, 105] employ the multivariate linear regression technique for a one-tier network. However, [105] focuses more on modeling inter-request dependencies of session-based systems and [76] focuses on mechanisms to deal with issues such as insignificant flows, collinear flows, space and temporal variations, and background noise.

3.1.1 Non-Linear Regression

If response time metrics are available, the service demands problem can be posed as a nonlinear least squares estimation. An example of research where regression splines are utilized is [30]. To find model parameters in nonlinear least squares problems, usually algorithms such as Levenberg-Marquardt [32, 99] can be used. In [32], for example, a weighted nonlinear regression is iteratively refined. Weights used in each iteration are based on observation residuals from the previous iteration. Points

that are outliers are down-weighted, and their influence on the fit is decreased. Iterations continue until the weights converge. Note that, the convergence of the parameters and the correctness of the estimation depend on the underlying model. If applying maximum likelihood estimation (MLE) on the model does not follow a convex form, it is quite possible that the regression does not converge, or converges to an incorrect value. In [104] and [65], multi-class queuing models were used in a two-tier web cluster, to infer service times per-class at different servers using throughput, utilization, and per-class response time measurements. They try to minimize the sum of predicted response time mean square errors using a non-linear optimization solver and a quadratic minimization program. [52] also uses a combination of MLE and the queuing model for estimation.

References [39, 40] claim to perform the demand prediction of enterprise workloads by discovering patterns, but the referred to demand is per-application not per-request; thus, they do not tackle the same problem as ours.

Despite simplicity and ease of implementation, the regression-based approaches have three major shortcomings:

1. In the presence of missing measurements, the regression problem might not be a convex optimization. For example, in the case where measurements are simply throughput and utilization, if a resource utilization or a class throughput is missing, terms such as $X_{c,t}.d_{c,k}$ might make the optimization

non-convex.

2. Even if the solution to the LSE problem can be formulated analytically (i.e. through algebraic operations on matrices), the computational complexity is still third-degree polynomial with respect to the number of samples T and resources K: $O((T.K)^3)$. Thus, as the size of the regression window grows (i.e. to provide more accuracy), solving for a large-scale environment (cloud) becomes more computationally expensive.

3. There is no way to specify the rate of change in the unknown variables explicitly. The value of unknown demands is considered constant within the regression interval. As a result, the number of samples within each steady-state interval determines how quickly the estimated demands are going to change with time. With fewer samples, they will change more frequently and with more samples they will change less.

3.1.2 Bayesian Approach

Papers [101, 102, 109] use Bayesian estimation to effectively deal with the problems of regression-based approaches enumerated earlier. They treat all unknown parameters as variables that vary over time. Thus, they can deal with the addition of a missing data measurement to unknown parameters without extra complexity. They

also reduce the amount of computation by solving the associated problem analytically and deriving a filter for it (i.e. Kalman type). This filter greatly reduces the estimation cost by doing step-by-step computation based only on the measurements of the current step and the state of the previous step, thus reducing computations to $O((C \cdot K)^3)$. Using a Kalman filter, they also specify the rate of change of the unknown parameters by specifying the process noise and the measurement noise covariance matrices.

The only issue in using the Bayesian approach is convergence. Convergence requires that at minimum, we have more measured parameters than estimated ones. As will be discussed in chapter 4 in detail, this convergence condition either puts a limit on the number of classes whose demands are to be estimated or requires a considerable number of parameters to be measured. The major contribution of our approach is dynamic clustering of user classes, to minimize the number of unknown variables. Yet, we leave enough classes to satisfy the accuracy criterion.

To the best of our knowledge, the combination of dynamic clustering and performance model parameter estimation for multi-class models has never been investigated. The only close reference to our work in terms of the final goal is [88]. In [88], Sharma et al. try to categorize requests based on resource usage characteristics. However, [88] ignores the prior knowledge about requests in the system and classifies the requests on-the-fly. Also, they use a machine learning technique called

Independent Component Analysis (ICA). Of course, ICA will alleviates the need for server instrumentation, but might affect the accuracy due to not using the extra available information (i.e. of the classes of users).

3.2 Application Resource Share Adjustment in Cloud Using Dynamic Empirical Models

Maintaining application level QoS guarantees has been the subject of much investigation. Recently satisfying the dual objectives of QoS guarantees and server consolidation in cloud computing environments has received considerable attention. Current approaches formulate the optimization problem in different forms.

One approach attempts to minimize cost subject to performance constraints [58, 59]. In this approach, the SLA represents constraints rather than flexible goals. Constraints could be based on response time or throughput. For example [59] finds the minimum cost deployment subject to processing capacity and user throughput constraints. It seeks deployments which minimize the overall cost of the hosts used, subject to meeting average delay and throughput constraints for each application as posed by its SLA.

The second approach attempts to minimize cost while maximizing QoS attributes simultaneously, through multi-objective optimization or MOO [57]. For

example, Pareto-optimal solutions can find a good trade-off between conflicting performance and cost-saving goals rather than finding a single global optimum [89]. Geometrically, these well-balanced solutions concentrate around the "knee" of a multi-objective curve.

The main difference between our approach and the related work is the use of dynamic empirical models (for each application) within the global optimization loop. These models update themselves at runtime in order to adapt to perturbations in the environment not captured in initial model specification. This results in more accurate models being passed to the optimizer, allowing for better resource utilization on a global scale.

3.3 Service Replica Placement Considering the Reconfiguration Cost

Static or one step ahead optimizations are the main tool to target minimization of infrastructure cost for private clouds while meeting applications QoS guarantees. But, when one takes into account the cost of reconfiguration, one has to add the time dimension into the optimization. This makes the optimization very complicated, and this complexity should be dealt with using the proper method. The approach we take here is using the Model Predictive Control (MPC) framework.

In terms of related work, there are only a couple of papers that target the same kind of problem. Paper [107] discusses the service placement in geographically distributed clouds through MPC. However, [107] defers from our approach in two ways: (i) it uses a single class open queuing network model, which is much simpler than the closed multi-class model we use, and (ii) it derives an analytical solution to a SSEC version of the problem first and then targets this solution as a desired state through MPC, considering the quadratic reconfiguration stage cost. In our case, this was not possible because we needed a sparse deployment at each step. From the same authors, paper [106] discusses controlling the optimal number of active servers for Google clusters where web applications and data processing jobs are both considered as generic jobs treated under scheduling policies. In this paper, they also use MPC, similar to our approach. They first found the optimal steady-state solution through a single class queuing model and then reached a solution considering the reconfiguration cost through MPC.

Other examples of using MPC in computing-resource management context are [11, 19, 23, 48, 77]; however, these examples do not target service placement. In [11, 19, 48] MPC is used for managing web server power consumption by changing the frequency at which the CPU is operating. In this example, the number of choices of frequencies is quite small (i.e. less than 10 alternatives), changes in the frequency are trivial, and the feedback does not have a lot of delay. They were able

to use a very small lookahead horizon and a simple model, and calculate the actual expected value of the cost over process noise (w_j) distribution, for every action path, without getting a state explosion. In [23] MPC is used to decide about the amount of data each node of a cluster should send through a set of data streams, and the amount to cache to disks. The proposed solution maximizes the throughput up to the network congestion point. In this paper the cost function was quadratic, which greatly aids with solving the MPC problem (heuristics not needed).

3.4 Summary

In this chapter, we discussed related work for the problems we introduced in chapter 1. The first problem we targeted is scaling up the EKF based estimation of service demands for large-scale service centres without increasing the monitoring overhead. Currently, there are two sets of techniques for service demand estimation: regression-based approaches and Bayesian filtering approaches. The Bayesian approach addresses the existing issues with regression-based approaches, such as the ability to tune the expected rate of parameter changes over time. However, the Bayesian approach suffers from a lack of convergence when the number of missing data items and hidden parameters exceeds the number of monitored metrics. Our proposal improves the Bayesian estimation in terms of convergence. It reduces the proportion of unknown parameters and metrics to measured ones by grouping user

classes, giving the filter a chance to converge with less monitoring overhead, and less computational complexity.

The second problem we target in this thesis is using dynamically tuning to improve the accuracy of the empirical models; we then trace this accuracy in the overall performance improvement in the allocation. We try to adjust the resource shares of a set of virtual machines that run on behalf of a set of applications. These VMs are deployed on a limited number of physical machines. The goal is to best satisfy the overall application performance, performance being described in terms of a set of utility functions. Our approach is based on point-wise stationary approximation (PSA) and solved for a set of the steps separately.

The third problem was the placement of services considering the reconfiguration cost. MPC is used to derive an optimal action at any time step based on the amount of workload to the services experienced to that point. This let us take the reconfiguration cost into account and minimize it.

4 Tracking Adaptive Performance Models using Dynamic Clustering of User Classes

This dissertation uses a model to project the state of the system under different actions. To do this projection, one needs to have the current state of the system in terms of the model elements. The objective of an estimator is to describe the system in terms of a given parametric model. This identification process includes finding or estimating the parameters of the model using the data obtained from the system, in an ongoing fashion.

In our case, the model corresponds to the cloud and the service instances, which run within the cloud. Thus, the identification mostly corresponds to the estimation of workload intensities (the number of users and the think times in a closed workload case and the arrival rate in an open workload case) and the service demands which are treated as hidden parameters of the model [30, 39, 40, 65, 76, 84, 85, 104, 105].

The estimation is carried out using the measured parameters of the system such as device utilizations ($U_{h,k}$), or class throughputs (X_c), and response times(R_c), if

available. For each measurement metric obtained from the monitoring subsystem, we have one or more relations according to the queuing theory. We use these relations in the context of the Kalman filtering approach.

4.1 Bayesian Estimation of Hidden LQN Parameters Using Extended Kalman Filter Estimator

In the Bayesian approach, the probability estimate for a hypothesis is updated as additional evidence is learned; this makes Bayesian approach a good fit for estimation tasks on sequential data. One can look at performance data as a sequence of measurements, where the unknown performance parameters are changing parameters to be estimated from observations [101, 102, 109].

All parameters are taken as latent variables. So it is assumed that the parameters distributions also vary over time, and there exists a *state-space model* that specifies the dynamics of latent variables and the relation among the latent variables and the observations. Assuming that one knows the state and the measurement noise covariance structures, one can estimate the model parameters by solving the associated optimization. Assuming Gaussian noise, subject to model and domain constraints, the estimation problem can be described using the following optimization expression: [18]

[18]Note that, determining the optimal parameter estimates cannot be done using a weighted

$$\underset{x_0,\ldots,x_T}{\text{minimize}}\ (x_0-\hat{x}_0)P^{-1}(x_0-\hat{x}_0)+\sum_{t'=0}^{T-1}w_t^TQ_t^{-1}w_t+\sum_{t'=0}^{T}v_t^TR_t^{-1}v_t$$

subject to:

$$\hat{x}_t=[\overrightarrow{\hat{d}_t},\overrightarrow{\hat{N}_t},\overrightarrow{\hat{Z}_t}]^T$$

$$u_t=[\overrightarrow{\tilde{d}_t},\overrightarrow{\tilde{N}_t},\overrightarrow{\tilde{Z}_t},\overrightarrow{\tilde{\Omega}_t},\overrightarrow{\tilde{\rho}_t}]^T$$

$$z_t=[\overrightarrow{R_t},\overrightarrow{U_{h,k',t'}},\overrightarrow{X_{s,h,t'}}]^T$$

$$x_{t+1}=x_t+w_t$$

$$z_t=LQM_{\tilde{x}_t}(\hat{x}_t)+v_t$$

$$d_{c,s,h,t}>0 \qquad \text{for each } c,s,h,t$$

$$0<U_{h,t}<1 \qquad \text{for each } h,\ k$$

where t denotes the discrete time. $\tilde{x}_t$ is used to represent the known portion of LQM parameters such as multiplicity of resources at time t. In other words, every value of $\tilde{x}_t$ defines a function of x_t, denoted by $LQM_{\tilde{x}_t}$. w_t is the process noise with a normal distribution, zero mean, and the covariance of Q_t (i.e. $N(0,Q_t)$), v_t is the observation noise with a normal distribution, zero mean, and the covariance of R_t (i.e. $N(0,R_t)$). x_t is the state to be estimated, composed of unobserved LQM parameters, with the mean of $\hat{x}_t$ and covariance matrix of P_t (i.e. $N(\hat{x}_t,P_t)$). z_t

---

least squares optimization of the measurement residuals (i.e. $y_t-h(x_t)$) with respect to the model parameters θ, since the process equation in the model is non-deterministic.

is the vector representing the measured performance. $LQM_{\tilde{x}_t}(\hat{x}_t)$ is, in this case, the observation model based on the queuing theory formulas 2.12 to 2.15 and it is non-linear with respect to the state vector.

The solution to the above optimization can be derived by an Extended Kalman Filter (EKF) [109], a variant of the Kalman filter [24].

Assumed prior knowledge of the filter includes distribution of the initial state, and process and measurement noise structures:

$$\begin{aligned} &\hat{x}_0 \\ &P_0 = E[(x - \hat{x}_0)(x - \hat{x}_0)^T] \\ &Q_t = E[w_t w_t^T] \\ &R_t = E[v_t v_t^T] \end{aligned} \tag{4.1}$$

where $\hat{x}_0$ is the initial estimate, P_0 is the initial error covariance matrix, and Q and R are the covariance matrices for the process and measurement noise, respectively.

The filter computations are recursive, beginning from an initial estimate $\hat{x}_0$, and an initial error covariance matrix P_0. Each recursive step can be summarized as follows:

1. The core filter calculation is the update of the state estimate $\hat{x}_t$ and error

covariance estimate P_t by the linear feedback equation:

$$H_t = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_t^-, u} \tag{4.2}$$

$$K_t = P_t^- H_t^T (H_t P_t^- H_t^T + V_t)^{-1} \tag{4.3}$$

$$e_t = z_t - LQM_{\tilde{x}_t}(\hat{x}_t^-) \tag{4.4}$$

$$\hat{x}_t = \hat{x}_t^- + K_t e_t \tag{4.5}$$

$$P_t = P_t^- - K_t H_t P_t^- \tag{4.6}$$

2. The filter then projects the state estimate $\hat{x}_t$ and the error covariance matrix P_t forward one step:

$$P_{t+1}^- = A P_t A^T + Q \tag{4.7}$$

$$\hat{x}_{t+1}^- = \hat{x}_t \tag{4.8}$$

Here, e_t denotes the prediction error vector obtained from the current modelled and observed output vectors ($LQM_u(\hat{x}_t^-)$ and z_t respectively), and P_t^- denotes the value of P_t at time t given measurements up to time $t-1$ (i.e. $P_{t|t-1}$). H_t is the matrix of sensitivity values or partial derivatives of the model function (i.e. $LQM_{\tilde{x}_t}(x_t)$) with respect to the parameters at their current values x_k, whose jth column is the derivative of $LQM_{\tilde{x}_t}(x)$ with respect to x_j. Essentially, this Extended Kalman filter (EKF) [25] linearizes $LQM_{\tilde{x}_t}(x_{t-1})$ by a first order Taylor series around the state estimate and does not take linearization errors into account.

Also note that here we assumed the process model chosen to describe the state is a "random walk." In this scheme the system state x_t (here demand and workload) is assumed to evolve due to random drift w as: $x_{k+1} = x_t + w_t$. This means that a minimal assumption is made during the estimation of hidden parameters of this model: that the covariance matrix for w_t is known (we denote it by Q_t). Usually the covariance matrix is diagonal since workload components (number of users and think time) and user demands are assumed to change independently. The values are sometimes set according to $Q_{i,i} = \alpha x_0(i)$, where $x_0(i)$ is the initial value of the i-th state variable to be estimated, and α is the ratio of expected changes for x_i's, (e.g. 0.02) with respect to their initial values.

The optimality and the convergence properties of the Kalman filter depend on the way the functions are linearized around the current estimate of x. The approximated H matrix and the equations of the filter depend on the following three factors:

The required estimates. The main estimates to be discovered are the service demands, and most of the time, the user population (N_c) and think time (Z_c) of each class. In this chapter, we assume that each service replica is identified by a subscript $j = 1...J$ which summarizes a tuple (s, h) (see the chapter 2). In other words, a replica j is implicitly associated with a host h and a service s. So a demand $d_{c,s,h,k}$ (introduced in the chapter 2) is re-written as $d_{c,j,k}$. $d_{c,j,k}$ represents

the service demand of a class c at a replica j, on a hardware resource k. This is done to associate the variables exclusively to the unknowns and reduce the total number of variables. If there is no replica of a service s placed on a host h, there will not be any variable $d_{c,j,k}$ defined for its estimated value. Note that the Kalman filter does not provide a way to enforce the equality constraints such as $d_{c,s,h,k} = 0$ during estimation. So, one needs to substitute such variables with the associated constants.

The measured parameters. Usually the throughput of each class of users (X_c) can be easily obtained. The mean utilization of each resource type k (e.g. CPU, disk, and network) on each host h ($U_{h,k}$) and the mean response time for each user class (R_c) are also usually available.

The missing data items. On certain occasions, a subset of metrics from a subset of servers might be missing. For example, it is usually not feasible to get response time metrics from individual service replicas ($R_{c,j}$) without code manipulation or instrumentation. These missing data items are also treated as variables in our model.

4.2 Dynamic Clustering of User Classes

An important issue in using the Kalman filtering approach is convergence. The necessary and sufficient condition is given by the identifiability condition [91, 95, 98].

The convergence requires that at minimum, we have more measured parameters than estimated state parameters: i.e.,

$$\dim(x) \leq \dim(y) \tag{4.9}$$

As we know, the estimated parameters might include: (i) the demand of each class, on each service replica, on each resource ($d_{c,j,k}$), (ii) the mean think time for each class (Z_c), (iii) the number of users in each class (N_c).

Thus $\dim(x)$, based on the number of classes (C), the types of resources available in each server (K), and the number of service replicas in the deployment (J) is as follows[19]:

$$\dim(x) = C \cdot J \cdot K + C + C = C \cdot (J \cdot K + 2)$$

In the case of $K = 1$ (i.e. only one resource type), we have:

$$\dim(x) = C \cdot (J + 2)$$

Note that in reality, the services can be classified into two kinds: the application-specific services, and the generic services. The relationship between classes and application specific services is a sparse one; meaning that there is a small number of application services used exclusively by each class of users. On the other hand, the relationship between the classes and the generic services is very dense because

[19]The dot sign was used to represent the multiplication.

almost all the classes make use of some primary core services. This sparsity and uniqueness, which constitutes an N-to-1 relationship between application services and classes, is a great help when it comes to estimation of the demands on user services. In a sparse deployment, the term $C \cdot J \cdot K$ will be hugely decreased due to the fact that not every class c makes use of a service replica j.

The measured parameters are most likely the mean response time and throughput of each class and the utilization of each server. If we also have a number of other measurements denoted by ϕ (for example the response time of each class at a service replica gives us another C measurements) it results into:

$$\dim(y) = \phi + 2C + H$$

and the identifiability condition 4.9 reduces to:

$$\phi + 2C + H \geq C \cdot (J + 2) \tag{4.10}$$

$$H + \phi \geq C \cdot J \tag{4.11}$$

$$C \leq H/J + \phi/J \tag{4.12}$$

thus $\dim(x) > \dim(y)$, if C is reduced to at least $(H + \phi)/J$. Note that replicas are at least equal to the utilized hosts ($H \leq J|U_h > 0$). So, the main factor is the number of extra measurements at the service replicas (i.e. ϕ). For each additional class beyond H/J, we need an additional J measurements. This linearly increases the monitoring overhead. We should, therefore, keep the number of classes small,

if possible.

In this section, we first discuss the modelling error due to clustering and then present an algorithm to determine the best choice of the number of clusters C and the grouping of classes into these clusters.

4.2.1 Modeling Error

Suppose the original C classes (i.e. $c = 1, \ldots, C$) are reduced to C' classes (i.e. $c' = 1, \ldots, C'$), and $c' = \psi(c)$ where ψ is the function that maps the class c in the original model to c' in the new model. Let $R_{\psi(c)}$ be the predicted mean response time of class $\psi(c)$ requests. For the case of no clustering (i.e., each class in the original model is treated as a separate class in the new model), let $R(C)_c$ be the mean measured response time of requests for the original class c. Then a modeling error measure $E(C')$ for a new model with C' classes is given by:

$$E_\psi(C') = \sqrt{\frac{1}{C}\sum_{c=1}^{C}\left(\frac{R(C)_c - R_{\psi(c)}}{R(C)_c}\right)^2} \tag{4.13}$$

The hypothesis is that the error $E(C')$ tends to decrease when the number of clusters is increased. However, finding the clusters and the multi-class performance model associated with the clusters is complex, as E is also a measure of how well the results from the filter and the model fit the measured data.

4.2.2 Dynamic Clustering Algorithm

Our classification algorithm is shown in Algorithm 1. Inputs to this algorithm are the LQM, a measurement vector z and an error threshold A. The vector z can include workload elements $\overrightarrow{\lambda_c}$ or $(\overrightarrow{N_c}, \overrightarrow{Z_c})$, the measured response time $\overrightarrow{R_c^{\mathrm{m}}}$ and throughputs $\overrightarrow{X_c^{\mathrm{m}}}$, and the total utilization of the servers $\overrightarrow{U_h}$. The algorithm generates the best values of the number of classes (i.e. C') for the new model, the grouping of original classes into these classes (i.e. function ψ), and the service demand estimates for the service replicas (i.e. $d_{c',j,k}$, $\forall j \in \{1,\ldots,J\}$, $\forall c' \in \{1,\ldots,C'\}$, $\forall k \in \{1,\ldots,k\}$).

In our investigation, the workload is time varying. The autonomic control loop is executed at regular intervals. If the modeling error of the existing grouping configuration is less than A, only the regular estimation is performed using the Kalman filter. On the other hand, if modelling error is greater than A, our clustering algorithm is invoked to obtain a new clustering of classes such that the modelling error becomes less than A (see algorithm 1).

The clustering algorithm starts by estimating the service demands of all the service replicas from the measured response times using the original model with no clustering (Step 1). The estimation at this step is done by using either an Extended Kalman filter or the linear or non-linear least squares methods described in chapter

Algorithm 1: The algorithm for estimation of service demands and clustering of user classes.

input : LQM, z, A

output : The best choice of the number of clusters C', aggregated demand $(d_{c',j,k})$ and mapping of old classes to new ones $c' = \psi(c)$.

1 $\{d_{c,j,k}\} \leftarrow$ Estimate $d_{c,j,k}$, the service demand of class c, service replica $j = (s, h)$, at resource type k, from measurement data ($\forall j = (s, h) \in \{1, \ldots, J\}$, $\forall c \in \{1, \ldots, C\}$, $\forall k \in \{1, \ldots, k\}$) using the model with no clustering; and set $C' = 1$.

2 **repeat**

3 $\quad \psi_{\{d_{c,j,k}\},C'} \leftarrow$ Cluster the services based on the $d_{c,j,k}$'s into C' clusters.

4 $\quad \{d_{c',j,k}\}, \{R_{c'}\} \leftarrow$ Estimate the parameters $d_{c',j,k}$, the service demand of class c' at replica j, at resource type k; solve the LQM with C' classes and obtain results for $R_{c'}$, the mean response time of class c' ($\forall j \in \{1, \ldots, J\}$, $\forall c' \in \{1, \ldots, C'\}$, $\forall k \in \{1, \ldots, k\}$).

5 $\quad E(C') \leftarrow$ Calculate the modelling error

6 $\quad$ Increase C' by 1

7 **until** $E(C') < A$;

8 Return C', ψ the original classes c' associated with each cluster c', and $d_{c',j,k}$ ($\forall j \in \{1, \ldots, J\}$, $\forall c' \in \{1, ..., C'\}$, $\forall k \in \{1, \ldots, k\}$)

2. Note that this is different from the estimation that is done at each step. The purpose of this estimation is solely for clustering the classes. If an EKF is used, the convergence criteria should be considered, and extra measurements should be used if necessary. One can use the least squares method, taking the demands constant over the past few steps (moving horizon estimation format or MHE) to avoid the need for extra measurements. The necessary condition for this case is to have the estimation horizon wide enough to contain enough data points for the least squares method.

In Step 3, the K-means clustering algorithm [49, 61] is used to perform an unsupervised grouping of the service demands. K-means has low complexity and is adaptable to the continuous nature of our problem. Moreover, it is able to detect clusters in an efficient way, which does not require computing the distance of all the points to one another. K-means takes as input the number of distinct clusters to generate (C') and will determine the size and members of the clusters (mapping between the classes and the clusters) based on the demands.

The modeling error $E(C')$ is calculated at Step 5 using the $R_{c'}$, that was obtained in Step 4. If $E(C') \leq A$, the algorithm terminates and returns (i) the best choice of the number of clusters, C', (ii) the classes assigned to each of the C classes, ψ, and (iii) the estimated service parameters for the different classes $d_{c',j,k}$ (see Step 7). If the modelling error $E(C')$ is larger than the acceptable error A (i.e. due to a

gradual change in service demands over time) the algorithm increases the number of clusters and performs another iteration. Steps 3, 4 and 5 are then repeated to compute $E(C')$ for the new number of clusters. Note that a larger number of clusters would increase the measurement overhead and the computational cost of estimation, but it will decrease $E(C')$.

4.3 Experiment 1: TPC-W benchmark and FIFA98 workload

To show the applicability of the method, we performed our first experiment on the estimation of service demands of the TPC-W benchmark [36] implementation. We deployed the Java implementation of TPC-W [9] with some modification, on a cluster of four Tomcat web servers and one single MySQL database server, with Linux as the operating system.

TPC-W is an e-commerce web application composed of 14 URLs, namely 'admin confirmation, admin request, bestsellers, buy confirm, buy request, registration, home, new product, order display, order inquiry, product detail, search request, search result, shopping cart'. Each of the URLs has a different service demands on web and database servers.

The application also comes with a workload generator, which takes as input:

(i) the number of users over time and (ii) a Customer Behavior Model Graph (CBMG), given to the workload generator as a transition probability matrix of a Markov chain. This workload is generated using Emulated Browsers (EB) whose behaviour and navigation is controlled by the Markov chain.

The number of users over time are derived from FIFA98's workload [14][20]. This workload reflects variations that servers might experience at runtime. We picked a portion of the day 21's workload (see Figure 6), extracted the web pages, lowered the number of requests by the factor of 2 (to factor in our smaller scale deployment topology), and finally used Little's law [63] (i.e. $N = X(R + Z)$) to convert the obtained throughput (X) to the number of users (N) and think time (Z) used by the emulated browsers of TPC-W. We assumed that the FIFA98 website had maintained the same response time over the sampling period.

For obtaining data, we monitored one of the web servers and the database and logged data collected from one of the web servers and the database; these included: response times, throughputs, and utilizations. Each sample represented a minute of work, and the total length of the experiment was 1 hour resulting into 60 samples.

Estimation. The application was modeled as two services, web and database.

[20]The reason we chose FIFA98's workload over some other workloads commonly used in computing systems performance community (e.g. Google Workload Traces [82]) is that the workload includes very detailed information about access patterns of a web application by online users. Basically the workload is the detailed log of Apache web servers. Google Workload Traces is mostly concerned with the amount of time it takes it Google cloud to complete a job and the performance of the scheduling. The trace also includes hardware level information about the resource usage on the physical machines.

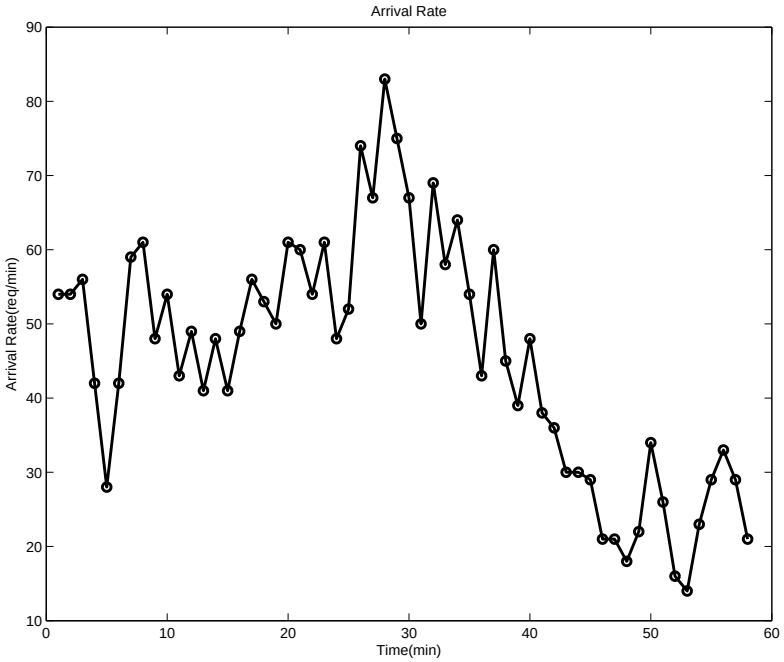


Figure 4.1: FIFA98 workload, day 21, over an hour.

Each service only has one replica placed on a separate server. We also took the users accessing the system through each individual URL as a class of users.

The average number of visits to web and database services are different for each class depending on the CBMG. This makes the average web and database service demands for each URL different, and subject to estimation. This leaves us with 28 parameters to estimate, 14 web service demands $d_{c,\mathrm{w}}$ and 14 database service demands $d_{c,\mathrm{db}}$. Note that there is only one service replica for each of the services in the system, each installed on a different host.

The monitored metrics include the throughput and response time of each class (28 measurements) and the utilization of each server (2 measurements). Since number of measurements is already more than the number of unknowns (28 versus 30), the convergence criterion is met even with no clustering.

Figure 4.2 represents the estimated service demands for 13 classes for the web service. One service demand is removed from the diagram to increase the clarity. Note that in the diagram the demands do not change frequently. We suspected that this is mainly because (i) the workload is unable to fully saturate the system or (ii) the combination of workload mixes are the same.

Our analysis on this experiment had three parts. First, we ran the algorithm statically with a different number of clusters and measured the modelling error. In our evaluation, we used an expanded definition of the modeling error metric given

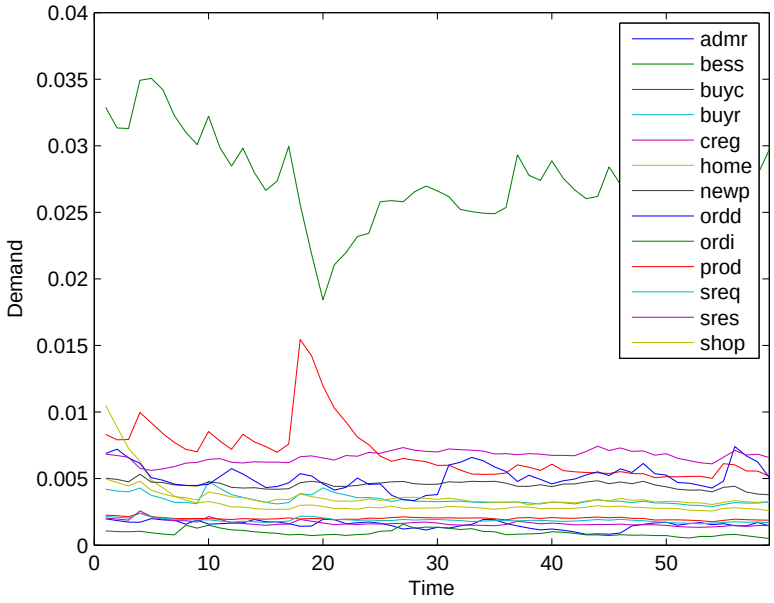


Figure 4.2: Estimated service demand for 14 URLs on Web server service.

by its average over the duration of the experiment (from 1 to T):

$$E = \frac{\sum_{t=1}^{T} E(C')_t}{T} \tag{4.14}$$

where t ranges over the estimation steps. $E(C)$ is the modelling error defined in equation 4.13 and T is the number of estimation steps.

As we expected, the error decreased while we increased the number of clusters (See Figure 4.3). The experiment also shows that modelling the system with one or even two classes introduces a large modelling error and that modelling with an intermediate number of classes (8, for example) might give us an acceptable modelling error.

In the second part, we applied our estimation and clustering algorithm to find

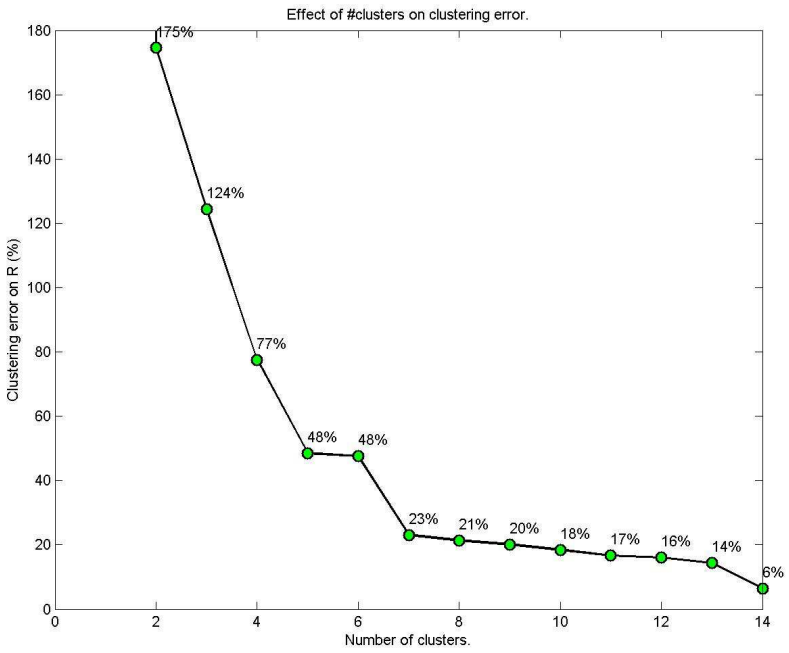


Figure 4.3: Modeling error decreases with the number of clusters.

the minimal number of needed clusters to reach a certain modelling error. The algorithm is applied at each sampling period and, as a result, the clusters change dynamically. As Figure 4.4 shows, we can reach 17% error using between 7 and 12 clusters for the duration of the experiment. For a modelling error less than 40% we need 9 clusters on average. In order to reach 5%, there are sampling periods in which we need maximum number of classes, which is 14.

In the third part of the analysis, we observed the correlation between the *within cluster sum-of-squares* (WCSS) for demands and the modelling error achieved using the estimation and clustering algorithm. We chose 140 different groupings. For each

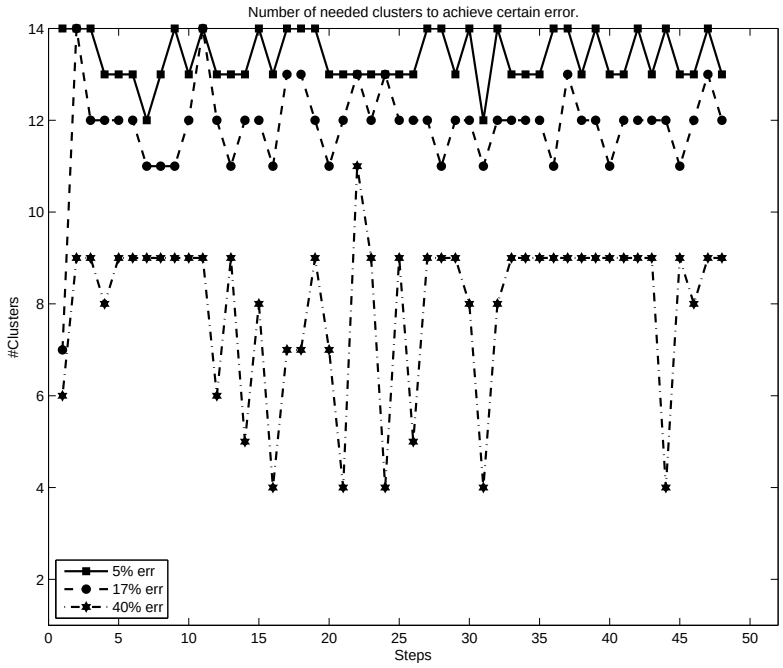


Figure 4.4: The minimal number of clusters needed to reach a specific modeling error threshold. We can reach 40% and 17% error, consecutively using 9 and 12 clusters on average. In order to reach 5% error, we have to perform full clustering.

number of clusters, we generated 10 random combinations. This let us navigate all possible WCSS's that could result from different groupings. Figure 4.5 shows that, on average we experienced a larger modelling error for the clusters with higher WCSS errors. In other words, the modelling error is minimized, whenever the WCSS is minimized. As a result, our assumption is validated since K-means is exactly the algorithm to minimize the WCSS.

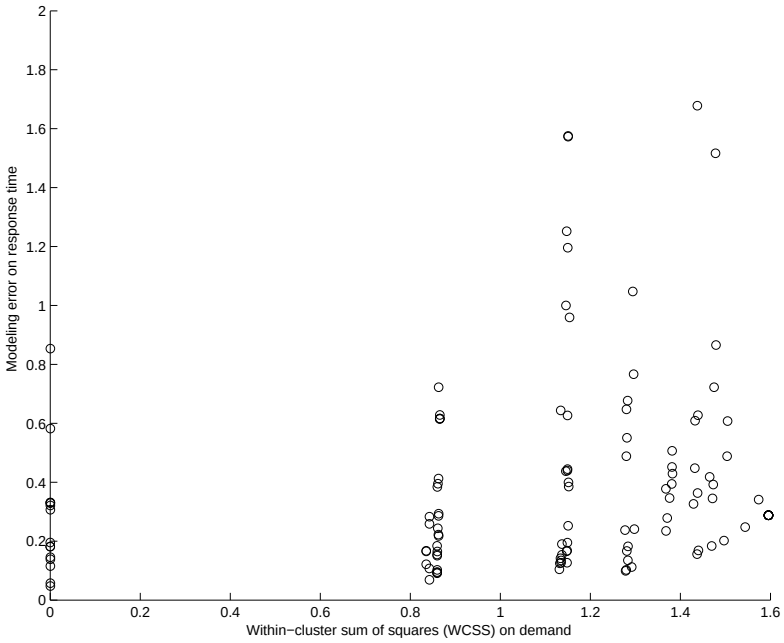


Figure 4.5: Correlation between the within cluster sum-of-squares (WCSS) for demands and the modeling error achieved on the response time's estimation.

4.4 Estimation and Clustering for highly variable demands

Our TPC-W experiment covered a short interval; so the service demands did not vary a lot. As a result, it was unlikely that a class moved from one cluster to another. However, in the long run (e.g. a month in real system measures), classes might change place due to variation in the CBMG (and consequently service visits and service demands).

A simulated experiment to investigate the effectiveness of the algorithm under non-uniform variations in demands was performed. This experiment showed the efficiency of the clustering and estimation for a web-based application where the estimated parameters change at different rates and phases. The simulator software used was the CSim discrete event-based simulator. To keep the presentation simple but also to highlight the merits of the proposed method, in the simulation, we varied the think time Z_c and the CPU demands, and kept N_c constant.

The system has 8 classes (c1-c8), two services (s1,s2), and two servers (w,d). Each service has only one replica, and each replica is placed on an individual server. Replicas are referred to as $j1$ and $j2$, or web and database.

The classes are associated with a number of users (N_c), a mean user think time (Z_c), and time variant service demands $d_{c,\mathrm{w}}$ and $d_{c,\mathrm{db}}$. The service demands follow a sine curve with the same period, but different phases (see Figure 4.6). Because

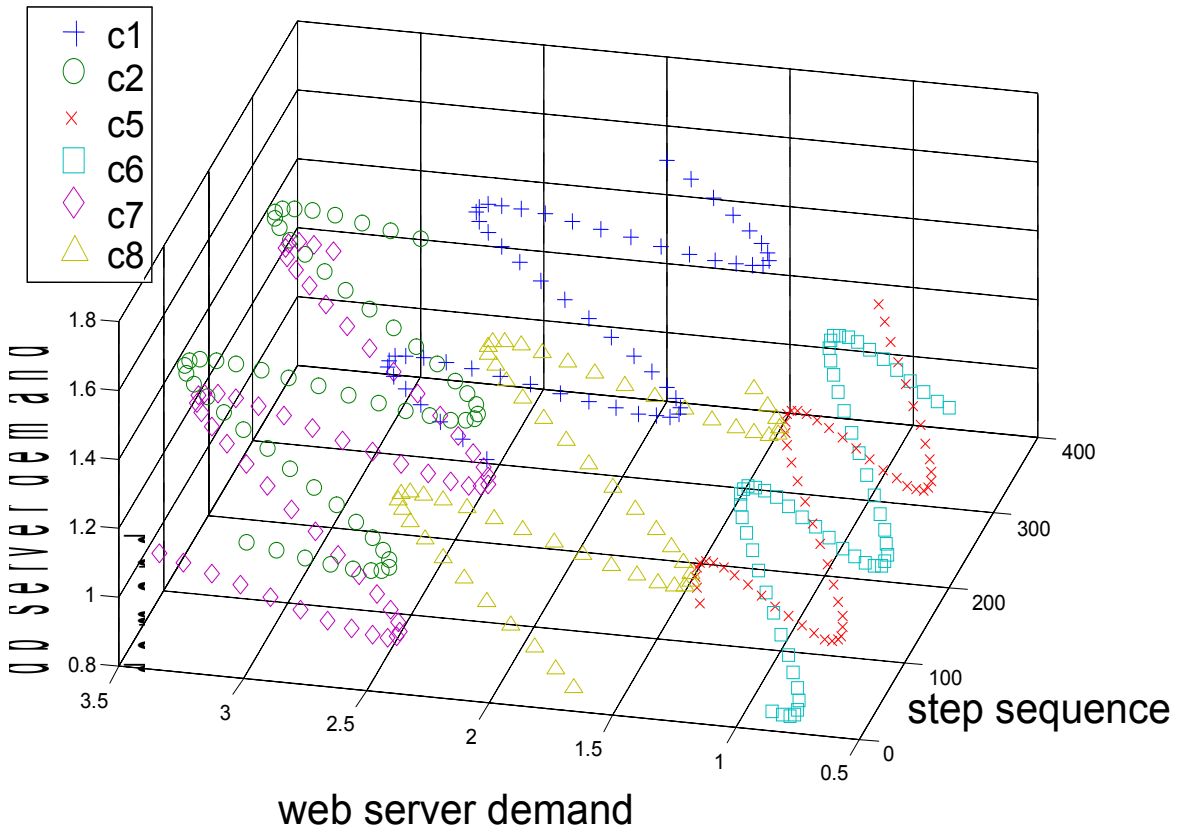


Figure 4.6: Service demands for 8 different classes [c1,c2,c5,c6,c7,c8] on two services.

of demand variations, classes migrate from one cluster to another, and the clusters evolve over time. In real applications, the service demands are not likely to change that dramatically, and we consider those variations as a stress load on the algorithm. Because of the variation of the service demands, we expect that the different classes will be re-clustered periodically.

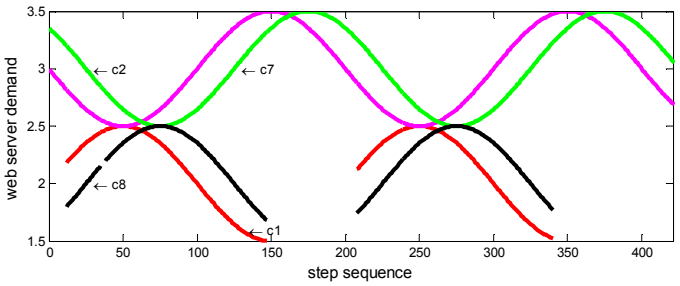
Table 4.1 shows the clusters suggested by the K-means algorithm. The acceptable modeling error A is set to 8% over 421 simulation steps. The column 'Action Nature' indicates the kind of change that has occurred when the past and current clusterings are compared. Ag, Br, Re, and Mv stand for aggregation, breaking, re-structuring, and movement respectively. The columns, "grouping (pre-event)" and "grouping (post-event)" represent the shape of the clusters before and after the re-clustering.

Figure 4.7(a) shows the variation of the service demands in a changing cluster ([c2,c7]+[c1,c8]). Real and tracked service demands for the cluster are shown in Figure 4.7(b) while the modelling error is depicted in Figure 4.7(c).

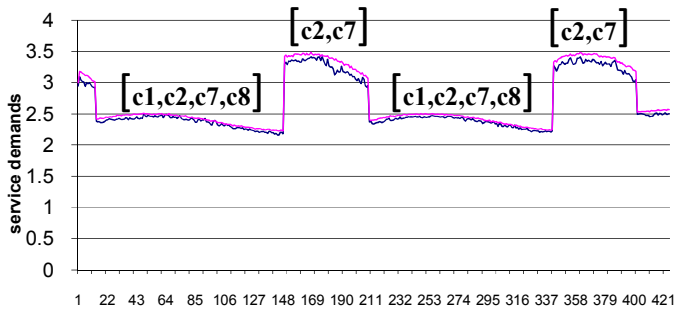
According to Figure 4.7(a), at step 50, $(d_{c1,\mathrm{w}}, d_{c1,\mathrm{d}})$ and $(d_{c2,\mathrm{w}}, d_{c2,\mathrm{d}})$ get close. This suggests that c1 and c2 should be in the same cluster near that step. At step 150, the service demands of c1 and c2 are quite different, indicating that these two classes are more likely in different clusters. These observations are consistent with our results where the clustering at step 50 is [c1,c2,c7,c8][c3][c4][c5,c6] and

Step	Action Nature	Grouping (Pre-event)	Grouping (Post-event)
12	Ag, Mv	[c1,c5,c6,c8] [c2,c7] [c3] [c4]	[c1,c2,c7,c8] [c3,c4] [c5,c6]
35	Br	[c1,c2,c7,c8] [c3,c4] [c5,c6]	[c1,c2,c7,c8] [c3] [c4] [c5,c6]
147	Mv	[c1,c2,c7,c8] [c3] [c4] [c5,c6]	[c1,c5,c6,c8] [c2,c7][c3][c4]
208	Ag, Mv	[c1,c5,c6,c8] [c2,c7] [c3] [c4]	[c1,c2,c7,c8] [c3,c4] [c5,c6]
239	Br	[c1,c2,c7,c8] [c3,c4] [c5,c6]	[c1,c2,c7,c8] [c3] [c4] [c5,c6]
340	Mv	[c1,c2,c7,c8] [c3] [c4] [c5,c6]	[c1,c5,c6,c8] [c2,c7][c3][c4]
400	Br, Mv	[c1,c5,c6,c8] [c2,c7] [c3] [c4]	[c1,c2,c7] [c5,c6,c8] [c3,c4]

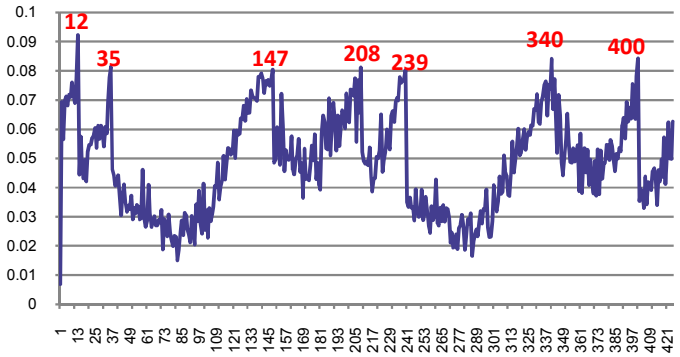
Table 4.1: Changes in clustering the structure of user classes due to service demand change.



(a)



(b)



(c)

Figure 4.7: Represents (a) the variation of service demands, (b) real and tracked service demands for 2 clusters [c2,c7] and [c2,c7,c1,c8], and (c) the modeling error with dynamic clustering applied.

[c1,c5,c6,c8][c2,c7] [c3][c4] at step 150. Note that c1 and c8 are shown in Figure 4.7(a) only when they are part of the cluster [c2,c7,...]. This explains the step-function-like behaviour of the estimated demand for the changing cluster in Figure 4.7(b).

Based on Figure 4.7(c), during the simulation the modelling error exceeds the threshold A, 7 times. This means the classification algorithm described in Section III.B is activated 7 times among the 421 steps. One can see that not every re-clustering necessarily results into a different number of clusters. For example, at step 147, the clustering changes from [c1,c2,c7,c8][c3][c4][c5,c6] to [c1,c5,c6,c8][c3][c4][c2,c7]. The number of clusters remains the same, but c1 and c8 have been moved to a different cluster. Figure 4.8 shows that the total number of clusters changes only 4 times over the 400 steps.

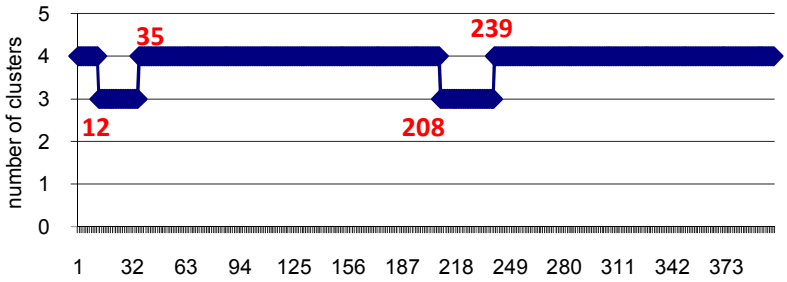


Figure 4.8: An estimation cases study: number of clusters over time.

We can conclude that our estimation and classification algorithm works quite well, since it is able to keep the error below A=0.08 with the smallest number of

clusters and acceptable frequency of re-clustering.

The algorithm also satisfies our claim: being able to exploit the complexity-accuracy trade-off and to keep both the modeling error and the monitoring and computational cost low. In terms of cost, our algorithm yielded half the required clusters (See Figure 4.8) compared to full classes estimation, which from the theory reduces the estimation cost by a factor 2^3 ($\frac{1}{2^3}$ of the original cost) and reduces the number of needed measurements by $4J$ (i.e. four fewer metrics for each replica). This is due to the fact that, the cost of the Kalman estimation in each step is dominated by Kalman gain calculation, which is $O(l^3)$ while l is the number of measured variables. The reason is that the dominating term during gain calculation is an inversion of a matrix of size $l \times l$:

$$K_n = P_n^- H_n^T (H_n P_n^- H_n^T + \ R_n)^{-1} \tag{4.15}$$

In this equation, R_n is measurement noise covariance matrix with size $l \times l$.

In terms of accuracy, it was able to keep the error near zero compared to the case of fully aggregated classes. See Figure 4.9 for the error comparison between our algorithm, the one cluster case and the full classes estimation.

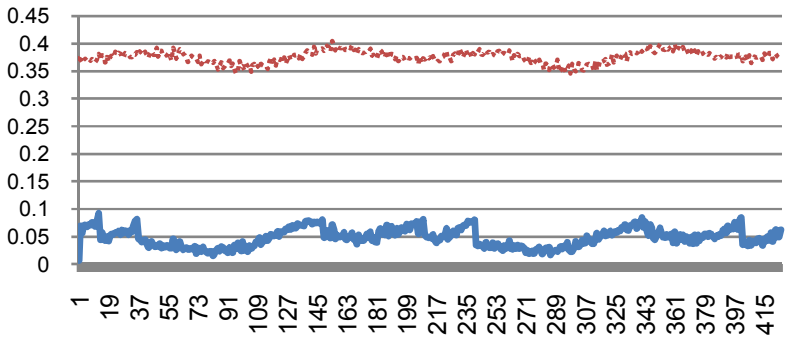


Figure 4.9: Comparison of the modeling error for one cluster case (dotted line), dynamic clustering case (solid line), and full URLs estimation (solid horizontal line at $y = 0$).

4.5 Summary

Using Kalman filtering and LQM for hidden performance parameters estimation, is a promising approach. However, it introduces lots of monitoring overhead when the number of classes is increased. To mitigate this, we investigated a tracking approach, that identifies the performance parameters of groups of classes (we call them clusters) instead of individual classes. We proposed an algorithm that finds the appropriate number of clusters with a pre-defined clustering accuracy.

We applied the clustering and tracking algorithm to two scenarios: (i) In the first experiment, we used our technique on the TPC-W benchmark deployed on a cluster of web servers. The workload was obtained from the well-known FIFA98 archives. In this experiment, first, we observed that the modelling error is reduced

as the number of clusters increases. Second, we tested 140 different random ways to cluster the classes and computed their average error values (E). We observed that a clustering with a smaller average distance of classes to centroids (i.e., with smaller cluster sum-of-squares) has less error. Thus, we showed the usefulness of the K-means algorithm. Moreover, we dynamically computed the number of needed clusters that keep the modelling error under a threshold. For example, if one can accept 17% error, the number of needed clusters for estimation would be dropped from 14 to 9 on average. (ii) In the second experiment, we deployed our algorithm on a system with 8 classes. It was shown that our Extended Kalman filter could track hidden states successfully, and the correctness of the filter rose as it tried more classes and re-estimated service demands.

Our algorithm also satisfies our claim: being able to exploit the complexity-accuracy trade-off and to keep both the modeling error and the monitoring and computational cost low.

In terms of cost, our algorithm yielded half the required clusters compared to full classes estimation, which from the theory reduces the estimation cost by a factor 2^3 ($\frac{1}{2^3}$ of the original cost).

5 Optimal Resource Share Adjustment in Cloud Using Dynamically Tuned Empirical Models

In order to allocate a set of limited resources to a set of applications optimally, there is a need for a model of the application service center. A portion of the related work on optimal deployment such as [58, 59] assumes that the service center follows an accurate first principle model. They suggest using a filter-based approach such as [110] to estimate the parameters of this model adaptively (i.e. through unsupervised learning). However, there is no guarantee that these first principle models accurately represent a service center. It is usually very hard to model all aspects of a service center such as software contention, concurrency, remote calls, a limited amount of memory, and a limited amount of network bandwidth.

In contrast to the first principle approach, an empirical approach can use a model obtained by regression analysis of application performances. The major problem with present empirical models, is the lack of tuning, based on actual measurement data (e.g. resource utilizations). When applications are deployed in a production

environment, the models start to lose accuracy. This inaccuracy is mainly because the production conditions are different from the test conditions [21].

Our contribution is to increase the accuracy of the empirical models by dynamic tuning using measurement data captured in real-time. We also trace the accuracy increase in the overall performance of the allocation. We investigate if a model built off-line through a nonlinear regression and dynamically tuned through an Extended Kalman Filter, can outperform a model which is not tuned. The other differences between our approach and previous work are in the use of the decomposed models, and in the use of a customized optimization routine to optimize the defined utility functions.

We show that static non-tuned application models result in a suboptimal allocation decision for some applications leading to missed SLAs. In contrast, we found that the use of dynamically tuned models resulted in more efficient resource allocations, closer achievement of SLAs and better utility.

The model we develop in this chapter corresponds to a virtualized private data center with limited resources. The unique feature of virtualized data centers when work conservation is enabled is that virtual machines isolate the applications. So, performance of each application can be modelled separately based on allocated resources. Each application's performance is modelled based on the amount of

[21] test conditions are used to build the training data sets.

resources allocated to its VMs. Since the simulator used in this chapter simulates applications with open workloads, we use open queuing network models as a basis of our model.

We assume that the cloud provider only tries to optimize the applications' performances by tuning the amount of resources allocated to the existing fixed set of VMs. Directly changing "relative share" of the allocated resource to each single VM can be done through the hypervisor's scheduler parameters. In addition, the assumption is that the cloud provider can monitor the performance metrics of the applications and thus build models based on the performance metrics and the given resources.

Our approach is presented in Figure 5.1. In this approach, for each application, the cloud provider maintains a dynamically updated performance model. A utility function based on a specific service level objective (e.g., response time) is also maintained for each application. The performance model for each application is updated periodically based on the measured performance attributes. The cloud provider then performs a system-wide global optimization (see Section 5.4) using the performance models and determines new resource allocations for each application for the following period.

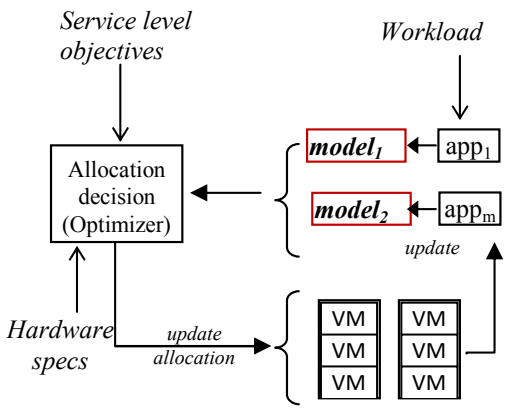


Figure 5.1: Feedback based optimization of resource shares.

5.1 General Definitions

In this section, we define the necessary notation used in the specification of the problem.

Applications: To denote applications we use the symbol c since one way to model applications is through classes as defined in a queuing model. Consider applications $c_1,\ldots,c_C$. Each application runs within one or more VMs and experiences a particular workload. For modeling purposes in this chapter, we assume each application experiences an open workload with certain inter-arrival time(s) denoted by τ_c. Also, we assume d_c is application c's hardware demand per application request (i.e. the number of CPU cycles for each request to be processed).

Hardware Structure: We assume a private service centre with heterogeneous resources composed of H hosts or physical machines (PM)s represented by the set

$h_0, ..., h_H$. We also assume each host has one resource type. The capacity of the resource of host h, is denoted by Ω_h.

Utility Function: Figure 5.2 represents a sample service level utility function, where the vertical line indicates the SLA target for the application and utility decreases as the value of the response time approaches the SLA limit. It is worth noting that any decreasing differentiable quasi-convex function can also be used as a service level utility function, and the only feature of this function is simplicity of the presentation in mathematical form. The shape of the function, especially after passing the SLA threshold, will affect the behaviour of the optimization algorithm.

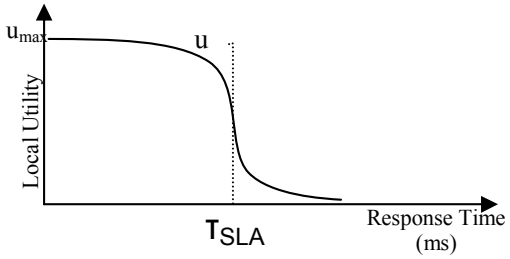


Figure 5.2: A smooth service level utility function; the vertical line indicates the service level objective of the application (as defined in the SLA).

Placement: We consider the VMs of applications which are already deployed on physical machines. We specify this deployment through a matrix ξ. For each application c and host h, if there is a VM allocated to the application on the host, the element (c, h) of matrix ξ is 1 and otherwise it is 0.

Resource Shares: We represent the allocation of VMs on PMs by a $C \times H$ matrix μ^θ. Each element $\mu^\theta_{c,h}$ of μ^θ denotes a resource allocation defining the percentage of the total resource (i.e. CPU) capacity of the PM h allocated to a running VM of application c.

5.2 Problem Formulation

Our objective is to maximize a global utility function (sum of application-provided resource-level utility functions u_c), subject to a set of capacity constraints which come from the physical layer of the private cloud. The optimization problem addressed here can be expressed as follows:

$$\begin{aligned}
&\text{given:} \xi_{c,h} \forall c \forall h, \tau_c \forall c, l_{\Omega,\sigma}, d_c \forall c \\
&\underset{\mu^\theta}{\text{maximize}} \sum_{c=0}^{C} u_c(R_c) \\
&\text{subject to:} \sum_{c=0}^{C} \mu^\theta_{c,h} < \Omega_h && \forall h \\
&\mu^\theta_{c,h} \geq 0 && \forall h \forall c \\
&R_c = l_{\Omega,\sigma}(\boldsymbol{\mu}^\theta_c, \tau_c, d_c) && \forall c \\
&\mu^\theta_{c,h} . \xi_{c,h} = \mu^\theta_{c,h} && \forall c \forall h
\end{aligned} \tag{5.1}$$

where d_c is application c's hardware demand per request and d_c's units is the number of CPU cycles.

l is a model that maps the application c's resource allocations (i.e. $\boldsymbol{\mu}^\theta_c$) to the

service level measure (i.e. response time) of the application, u_c is the local utility function for the application c, $\mu^{\theta}_{c,h}$ represents the allocation of a physical machine h to application c. In a case study we use a simulator that includes the number of CPU cycles as an input parameter. Each $\mu^{\theta}_{c,h}$ will be considered to have the unit of million instruction per-second (MIPS). $\boldsymbol{\mu}^{\theta}_{c}$ represents the vector of resources allocated to an application c on different hosts. The equality constraint $\mu^{\theta}_{c,h}.\xi_{c,h} = \mu^{\theta}_{c,h}\ \forall c \forall h$ implies that the only elements of $\mu^{\theta}_{c,h}$ that need to be optimized are those whose corresponding element in ξ is 1. Other elements of θ will be considered 0. It is an implication of the first and second inequality constraints that each allocation $\mu^{\theta}_{c,h}$ is constrained to lie in the interval $[0, \Omega_h]$ meaning that an application can maximally get the whole capacity of a PM. The equality constraint relates the response time of each application to the workload intensity, hardware demands, and the given capacities.

Note that the proposed problem solution has a best effort nature, and we treat a service level objective (i.e. target on a specific QoS metric) as a soft constraint by incorporating it into the objective function.

5.3 Selecting an Empirical Model

A performance model can be used to relate the physical layer specification of a customer's application (e.g. it's given resource shares) and its associated workload

to a service level measure (e.g. response time) quantitatively. We assume that the response time of each application c, can be modelled by the given resources (to VM's) of application c (i.e.$\boldsymbol{\mu}_c^\theta = \{\mu_{c,1}^\theta, \ldots, \mu_{c,H}^\theta\}$), the demand of application on the resource types (i.e. here only CPU d_c) and the workload of the applications (i.e. τ_c) using the function l:

$$R_c = l_{\Omega,\sigma}(\boldsymbol{\mu}_c^\theta, \tau_c, d_c) \tag{5.2}$$

where $\boldsymbol{\mu}_c^\theta$ is a vector of elements $\mu_{c,h}^\theta$, which represent the number of allocated CPU cycles (i.e., measured in MIPS), τ_c denotes the request inter-arrival time in seconds associated with the workload of the application c, d_c is application c's hardware demand per request (i.e. the number of CPU cycles for each request to be processed), and R_c is the average response time of the application.

The nonlinear model is further simplified by using the aggregate resource capacity measure $\mu_c^{\theta,\text{Agg}} = \sum_{h=1\ldots H} \mu_{c,h}^\theta$ instead of the vector of individual values $(\mu_{c,1}^\theta, \mu_{c,2}^\theta, \ldots, \mu_{c,H}^\theta)$:We will assume that all applications deployed on the infrastructure adhere to the same performance model, differing only in the setting of various configurable parameters.

Focusing only on one application called c, let $\boldsymbol{\beta}_{c,t}$ be a vector representing the model coefficients and $\left[\mu_{c,t}^{\theta,\text{Agg}}, \tau_{c,t}, d_{c,t}\right]$ be the measured inputs of the system (i.e. capacities, inter-arrival time and demand) at step t. The model maps

$\left[\mu_{c,t}^{\theta,\text{Agg}}, \tau_{c,t}, d_{c,t}\right]$ and $\boldsymbol{\beta}_{c,t}$ to the modelled output $R_{c,t}$:

$$R_{c,t} = f(\left[\mu_{c,t}^{\theta,\text{Agg}}, \tau_{c,t}, d_{c,t}\right], \boldsymbol{\beta}_{c,t}) \tag{5.3}$$

where f is a runtime approximation of the service level function (or performance model) l defined earlier.

While the function f can take several forms, the online-estimation techniques and optimization method are general and will be considered next in Subsection 5.3.1 and Section 5.4.

We considered three parametric forms for function f:

1. A simple linear model only containing predictor variables[22]:

   $f(\left[\mu_c^{\theta,\text{Agg}}, \tau_c, d_c\right], \boldsymbol{\beta}_c) = \beta_{c,1} + \beta_{c,2} d_c - \beta_{c,3}\tau_c - \beta_{c,4}\mu_c^{\theta,\text{Agg}}$

2. A more complex linear model with interaction terms [23] $\mu_c^{\theta,\text{Agg}}\tau_c$ and $\mu_c^{\theta,\text{Agg}} d_c$ to capture the effect of $\mu_c^{\theta,\text{Agg}}$ on response time at different inter-arrival times and demands:

   $f(\left[\mu_c^{\theta,\text{Agg}}, \tau_c, d_c\right], \boldsymbol{\beta}_c) = \beta_{c,1} + \beta_{c,2} d_c - \beta_{c,3}\tau_c - \beta_{c,4}\mu_c^{\theta,\text{Agg}} - \beta_{c,5}\mu_c^{\theta,\text{Agg}} d_c + \beta_{c,6}\mu_c^{\theta,\text{Agg}}\tau_c$

3. A non-linear model derived from the mean value based formula for open queuing networks ($R = D/(1 - \lambda D)$) [54] extended to account for capacity:

[22] The model is linear in the parameters $\beta_{c,i}$ not predictor variables.

[23] The interaction term is the product of the subset of our original predictors.

$$f(\left[\mu_c^{\theta,\text{Agg}}, \tau_c, d_c\right], \boldsymbol{\beta}_c) = \frac{\beta_{c,1} d_c}{\mu_c^{\theta,\text{Agg}} - \beta_{c,2} d_c/\tau_c}$$ [24]

To investigate the accuracy of the models, a data-set of the sample application's performance was generated synthetically by invoking the CloudSim simulator [26] using random values for the capacity, demand and the inter-arrival time. Figure 5.3 presents a partial visualization of the data-set. A regression analysis was then performed on the resultant data for each model, allowing confidence intervals to be computed for all model parameters. Usually, the use of a predictor variable is meaningful if its estimated coefficient's confidence interval is far from zero[25]. Moreover, a wide confidence interval usually means lack of accuracy and the need for more data. We formed approximate 95% confidence intervals for each $\beta_{c,i}$ using mean, standard error, and the normality assumption, to indicate the accuracy and significance of each $\beta_{c,i}$. The approximate 95% confidence intervals for coefficients were quite narrow, and the margins of error were orders of magnitude smaller than the coefficient. This result implies that the amount of data for modeling was adequate. However, when applying the regression analysis using the linear models, the intervals were very near to zero and coefficients were not significant. One possible explanation for the lack of significance is that only the fixed portion

[24]This equation can be derived by substituting D_c with $d_c/\mu_c^{\theta,\text{Agg}}$ and τ_c with $1/\tau_c$ from the base formula, and adding model coefficients (i.e. $\beta_{c,1}$ and $\beta_{c,2}$) to compensate for the differences between the system and a pure queuing model. The substitution requires the inter-arrival time (τ) to be a homogeneous Poisson process, and inter-arrival times (τ) to be exponentially distributed with parameter τ (mean $1/\tau$).

[25]There is no proven way for finding if the interval is far enough from 0.

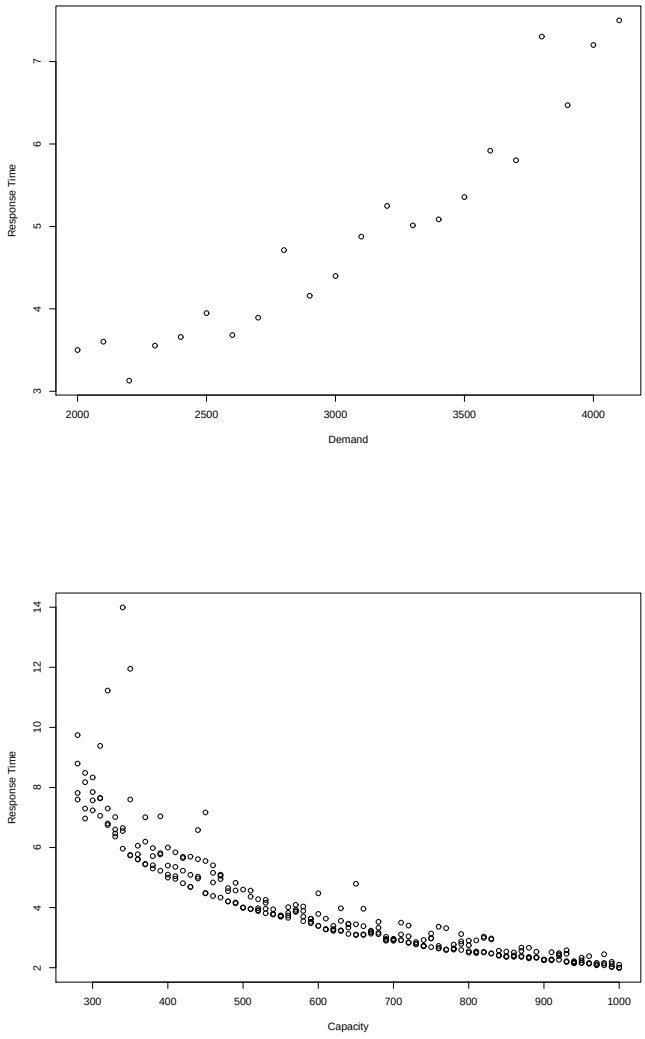


Figure 5.3: Visualization of pairwise relation between input parameters (capacity,demand) and the response attribute (i.e. response time) for a simulated application running on a single VM.

of the response time (i.e. delay at server) was accounted for in linear models, not the queuing delay which might grow exponentially. The estimated coefficients for the nonlinear model were relatively significant. As the result of this analysis, the non-linear model's use was validated.

5.3.1 Online Estimation

Let $z_{c,t}$ be the measured system output (e.g. response time) at step t. We further assume that $\beta_{c,t}$ changes according to a random walk:

$$\boldsymbol{\beta}_{c,t+1} = \boldsymbol{\beta}_{c,t} + w_{c,t} \tag{5.4}$$

where $w_{c,t}$ represents some Gaussian noise. A filter maintains the estimate of $\boldsymbol{\beta}_{c,t}$ and updates it using the linear feedback equation based on new measurements $\left[\mu_{c,t}^{\theta,\text{Agg}}, \tau_{c,t}, d_{c,t}\right]$ and $z_{c,t}$:

$$\boldsymbol{\beta}_{c,t} = \boldsymbol{\beta}_{c,t-1} + K_{c,t}(z_{c,t} - R_{c,t}) \tag{5.5}$$

where t denotes a discrete time index, $K_{c,t}$, the dynamically computed "Kalman Gain" matrix and $z_{c,t} - R_{c,t}$ denotes the prediction error[26]. With assumptions of system linearity and Gaussian noise, the calculated Kalman gain $K_{c,t}$, is guaranteed to minimize the estimation errors for $\boldsymbol{\beta}_{c,t}$. The updated state will result in a dynamic model $f(\left[\mu_{c,t}^{\theta,\text{Agg}}, \tau_{c,t}, d_{c,t}\right], \boldsymbol{\beta}_{c,t})$ that will be accessed multiple times during

[26]Note that $R_{c,t}$ was calculated using equation 5.3.

optimization cycles. Function f and its non-measurable and measurable parameters can have many representations (as shown in Section 5.5).

5.4 Optimization

In this section, the utility maximization problem defined in equation 5.1 is solved. Algorithm 2 represents a centralized solution for this problem using the primal method.

The input to the algorithm includes the initial capacities (i.e., μ_0^θ), the number of iterations during optimization ($k_{\max}$), the performance model ($f_{\boldsymbol{\beta}_{c,t}}$) and the utility model u_c for each application. The output of the algorithm is the optimal allocation matrix (μ_{opt}^θ), and maximum utility gained from that allocation (fp_{best}) that is obtained iteratively using $k_{\max}$ iterations.)

The algorithm performs $k_{\max}$ iterations. In each iteration it does the following:

1. Computes the constraint values (a constraint is violated if its value is less than 0). A constraint for each physical machine (PM) is the sum of all resources given to VMs deployed on that PM minus the PM's total capacity (step 1).

2. If the current allocation point is feasible (i.e. no constraint is violated) the algorithm re-calculates the objective function value U_0 using the current allocation ($\mu^{\theta,\text{Agg}}$), performance model f from equation 5.2, and utility model

Algorithm 2: A centralized solution for the problem using primal method.

input : initial capacities μ_0^θ, maximum number of iterations $k_{\max}$, performance model f, utility model u_c forall c, model coefficients $\boldsymbol{\beta}_{c,t}$ for all c

output : optimal allocation μ_{opt}^θ, Maximum utility fp_{best}

initialize: $fp = -\infty$, $fp_{best} = -\infty$, $\mu^\theta = \mu_0^\theta$, $k = 0$

while $k < k_{max}$ **do**

  compute constraint value U_i for each PM: $U_i(\mu^\theta) = \left(\sum_{c=0}^{C} \mu_{c,h}^\theta\right) - \Omega_h$

  if *no constraint is violated:* $max(U_i(\mu^\theta)) > 0$ *//move towards optimum:* **then**

    compute global utility function: $U_0 = \sum_{c=0}^{C} u_c(f(\left[\mu_{c,t}^{\theta,\text{Agg}}, \tau_{c,t}, d_{c,t}\right], \boldsymbol{\beta}_{c,t}))$

    record the global maxima: $fp_{\text{best}} \leftarrow max(U_0(\mu^\theta), fp_{\text{best}})$

    calculate objective function's subgradient: $g = \partial U_0(\mu^\theta)/\partial \mu^\theta$

    move towards subgradient and the optimum: $\mu^\theta \leftarrow \mu^\theta + \alpha g_0$

  else if *there is some violated constraint* **then**

    find most violated constraint using equation: $m \leftarrow argmax_i(\|U_i(\mu^\theta)\|)$

    compute the subgradient value of most violated constraint as follows:

      for each c, if $\xi_{m,c} = 1$ then $g_{m,c} \leftarrow 1$ else $g_{m,c} \leftarrow 0$

      for each h and c, if $h \neq m$ then $g_{h,c} \leftarrow 0$

    select step size: $\alpha = (U_i(\mu_{c,h}^\theta) + \epsilon)/(\|g\|_2^2)$

    move away from subgradient and the optimum: using equation 6. $\mu^\theta \leftarrow \mu^\theta - \alpha g_i$

  project each individual variable based on its local constraint:

  $\mu_{c,h}^\theta = min(max(\mu_{c,h}^\theta, 0), \Omega_h)\ \forall h \forall j$

  $k = k + 1$;

$\mu_{\text{opt}}^\theta = \mu_{c,h}^\theta$;

u_j defined by the administrator (step 5).

3. If the objective value was greater than the currently reached maxima, it would be recorded (step 6).

4. Then the algorithm calculates the objective subgradient using numerical differentiation (i.e. solving the current performance model multiple times for near values of the current point) as if the problem were unconstrained (step 7).

5. Finally, a new allocation matrix is calculated, by moving towards the objective function subgradient with a fixed step size (optimality update, step 8).

6. If the current allocation point is infeasible the algorithm chooses the most violated constraint[27] (step 10).

7. Then the algorithm projects the current point onto the set (half-space) of points that satisfy that inequality constraint. Projection is done by moving towards the opposite direction of the constraint function subgradient, g_i (step 15). The sub-steps 12 and 13 assert that, since each constraint is sum of allocated capacity for each VM, $\partial U_i(x)/\partial x$ is a matrix with same dimensions as x (with 1 in each element that affects the constraint). For example if $f(x)_i = x_1 - 6$ then $\partial f(x)_i/\partial[x1, x2] = [1, 0]$.

[27] Actually, this can be any violated constraint.

8. The step size is calculated in step 14. While $\epsilon > 0$ is a tolerance which can take-on different values (e.g. 10^{-3},10^{-2}, etc.).

9. Finally we check that each allocated capacity is feasible and longer than the local bounds (step 16): where lb is the lower bound and ub is the upper bounds.

For clarity, the equations referenced by the algorithm are listed in Table 5.1. To

1	$U_i(A) = \left(\sum_{c=0}^{C} \mu_{c,h}^{\theta}\right) - \Omega_h$
2	$U_0 = \sum_{c=0}^{C} u_c(f(\left[\mu_{c,t}^{\theta,\text{Agg}}, \tau_{c,t}, d_{c,t}\right], \boldsymbol{\beta}_{c,t}))$
3	$fp_{\text{best}} \leftarrow max(U_0(\mu^{\theta}), fp_{\text{best}})$
4	$g = \partial U_0(\mu^{\theta})/\partial \mu^{\theta}$
5	$\mu^{\theta} \leftarrow \mu^{\theta} + \alpha g_0$
6	$\mu^{\theta} \leftarrow \mu^{\theta} - \alpha g_i$
7	$i \leftarrow argmax_i(\|U_i(\mu^{\theta})\|)$
8	$[(k = i \wedge \xi_{k,c} = 1) \rightarrow (g_i)_{k,c} = 1] \wedge [k \neq i \rightarrow (g_i)_{k,c} = 0]\ \forall k \forall j$
9	$\alpha = (U_i(A) + \epsilon)/(\|g\|_2^2)$
10	$\mu_{c,h}^{\theta} = min(max(\mu_{c,h}^{\theta}, 0), \Omega_h)\ \forall h \forall c$

Table 5.1: Equations referenced by Algorithm 2.

describe the equations briefly: Equation 1 states that the value of the constraint[28] for each PM_i $(i > 0)$ is the sum of all resources given to VMs deployed on that PM minus the PM's total capacity. Equation 2 gives the global utility function U_0. Formula 3 is the incremental recording of objective value. Equation 4 is used to calculate the objective subgradient as if the problem were unconstrained. Equation 5 indicates moving towards the objective function subgradient with a fixed step size (i.e. optimality update). Equation 6 is projecting the current point onto the set of points that satisfy the inequality constraint i. Equation 7 denotes choosing the most violated constraint. Equation 8 is calculating the subgradient g_i of a constraint i (i.e. $\partial U_i(\mu^\theta)/\partial\mu^\theta$) while $i > 0$. Equation 9 calculates the step size in the feasibility update, while $\epsilon > 0$ is a tolerance and set to 10^{-3}. Equation 10 checks that each allocated capacity is feasible and is between the local bounds.

To illustrate the optimization algorithm, we consider a pre-built model of one physical machine hosting two applications, each running on a single VM. Both applications have CPU demands of 5100 and 4100 MIPS per request respectively, and response time thresholds of 16 and 7 respectively. The request inter-arrival time of both applications is 22 seconds, and their $u_{\max}$ is 10. A single PM's CPU, which is used to host these applications, has a total capacity of 1200 MIPS. Assumed capacities μ^θ_{11} and μ^θ_{21} are the CPU allocation for VM1 and VM2. Figure 5.4

[28]Value of a constraint here refers to the degree that the constraint is satisfied (positive values) or violated (negative values).

shows the entire configuration space over these capacity allocations, constrained by inequalities: $\mu_{11}^{\theta} + \mu_{21}^{\theta} < 1200$, $\mu_{11}^{\theta} > 280$, and $\mu_{21}^{\theta} > 280$. Note how the optimizer climbs up the global utility function, hits the constraint (i.e., $\mu_{11}^{\theta} + \mu_{21}^{\theta} < 1200$) then continues to climb and reach the maximum.

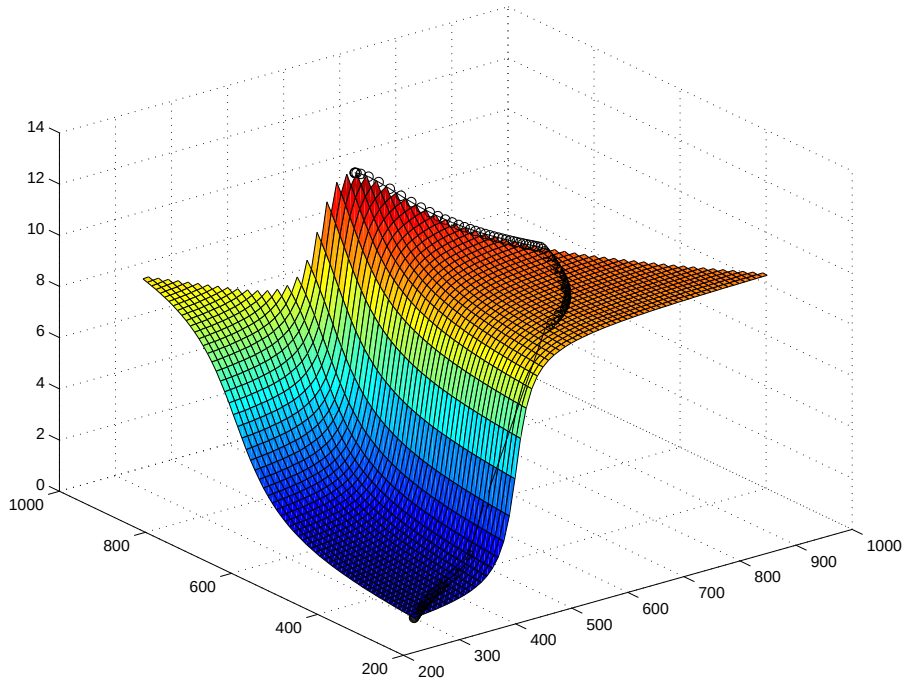


Figure 5.4: One sample run of subgradient optimization algorithm, finding the maximum utility using variations in capacity.

5.5 Case Studies

For the case studies, we simulated a private cloud with the CloudSim [26] simulator. We further considered a very simple local utility function for applications:

$$u_c = u_{\max_c} \cdot \frac{(\arctan(r_{\text{thr}_c} - r_c) + \pi/2)}{\pi} \tag{5.6}$$

where $u_{\max_c}$ denotes the maximum possible utility, r_{thr_c} denotes the SLA limit (in terms of the minimum response time) and r_c denotes the actual response time.

5.5.1 Case Study One

Consider a small cloud configuration with three applications (`app1`, `app2`, `app3`) deployed on two PMs with the following placement matrix:

$$\xi^T = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

That is, `app1` is deployed on PM1, `app3` is deployed on PM2, and `app2` uses both PMs by having a VM on each.

Both of the PMs have single core CPUs with each core having the processing speed of 1200 MIPS and 2 GB of ram. The VMs for applications are identical, each with image size of 1 GB, ram of 512 MB, and bandwidth of 1 GB.

The CPU resource share for each VM is initialized to 280 MIPS but will be adjusted by the optimizer. The applications have workloads with mean CPU de-

mand per request of 4100, 5700 and 500 MIPS respectively. For `app2`, the load is distributed evenly between the two VMs. We chose the arrival-rates of applications, from the FIFA '98 workload [14]. The inter-arrival times of requests for all dynamic pages were extracted, on a per-minute basis, from the first six hours of day 21 of FIFA98 workload. To incorporate the time-of-a-day effect, we divided the workload into three 2-hour periods and applied each as an application workload (see Figure 5.5). Moreover, we multiplied inter-arrival times of applications by 14, 11 and 8 respectively. The third application is more transactional in nature (i.e. lower demand and higher inter-arrival time) while the first two are characterized as batch-like.

The utility functions are defined for each application based on equation 5.6 with the following parameters for each application:

$$\mathbf{R}^{\mathrm{SLA}} = [15, 16, 2]$$

$$\mathbf{u}_{\max} = [20, 10, 5]$$

For each of 120 samples of inter-arrival times, we submit 20 transactions for each application in the simulator. We ran the experiment twice. One run used the static model with the coefficients computed off line as mentioned above. The second run used the same model but tuned at runtime. In the case of the tuned model, the new measurements obtained from the simulator (e.g. response times) were used to

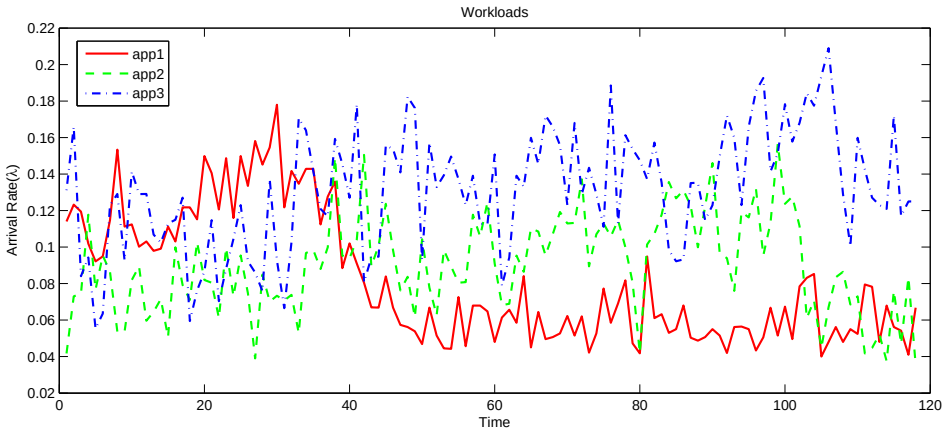
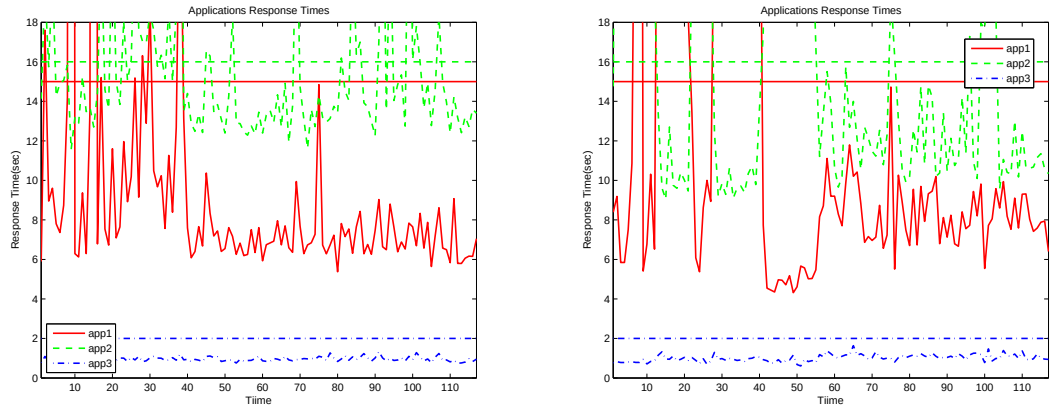


Figure 5.5: The workload of applications.

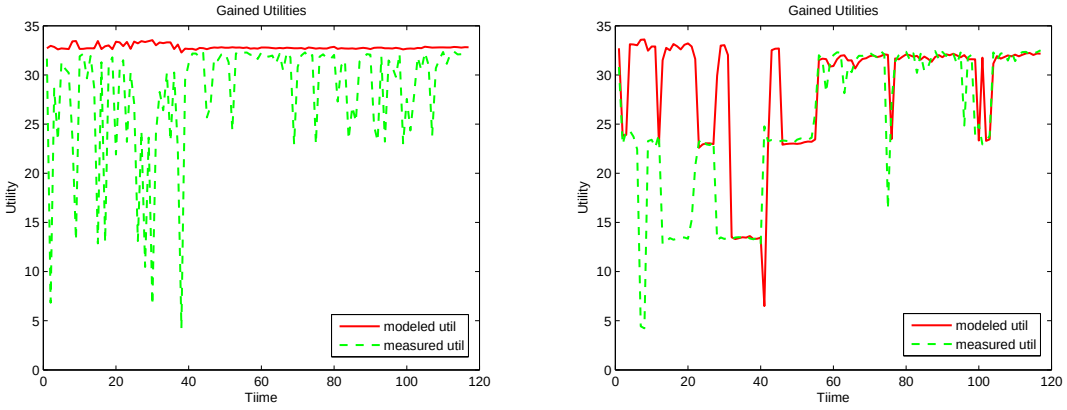
update the dynamic model's coefficients. In the case of a static model, the optimizer used the model obtained through regression analysis as-is and no tuning was performed. The optimizer then recalculated the new resource allocations, passed the new allocation vector to actuators to apply it to the system.

Figure 5.8 presents the results. The use of static non-tuned models by the applications results in a bad allocation decision for `app2` and leads to failure in meeting app2's SLA (see Figure 5.6a). This results in achieving a suboptimal utility (see Figure5.7a). In contrast, using dynamic models results in more efficient resource allocations being made, better commitment to SLAs for `app2` (see Figure 5.6b) and better utility (Figure5.7b). Figure 5.8a and 5.8b represent the allocated capacity to applications over time in case of dynamic models; 5.8a is for PM1 and 5.8b is for PM2.



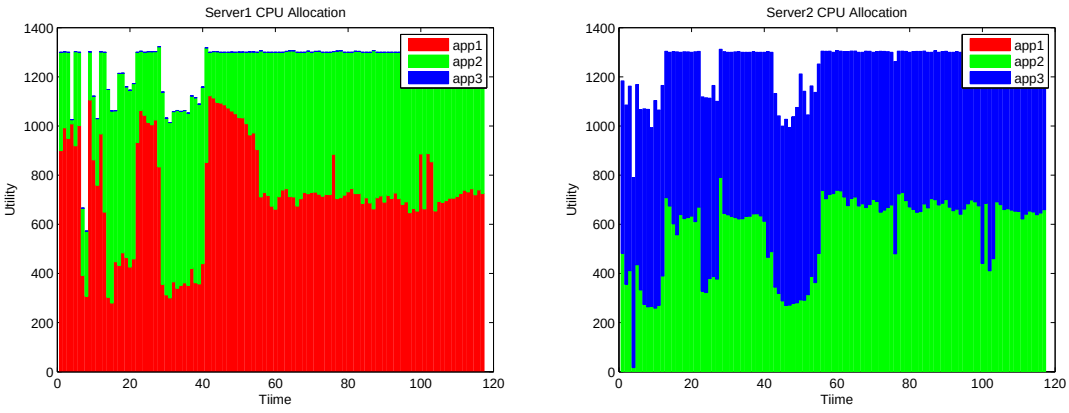
(a) (b)

Figure 5.6: The response time of applications together with SLA response times for (a) static and (b) dynamic models.



(a) (b)

Figure 5.7: The measured and modelled gained global utility using (a) static and (b) dynamic models over time.

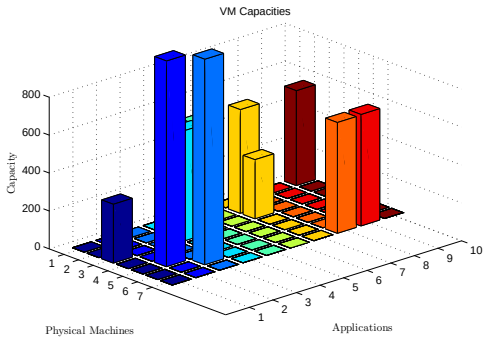


(a) (b)

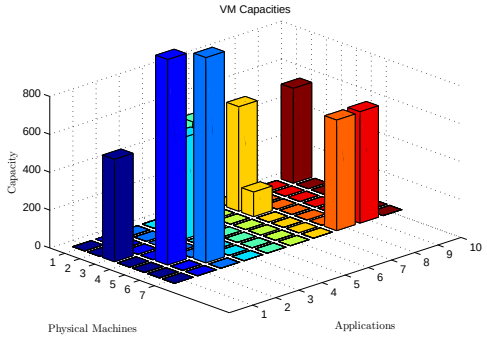
Figure 5.8: The allocated capacity to applications over time on (a) server one and (b) server two, using dynamic models.

5.5.2 Case Study Two

The second case study was performed to demonstrate the ability of our approach to update the resource shares in a way that compensates for the additional workload. Ten applications were deployed on seven PMs. VMs were assigned randomly to PMs. All application workloads were set to have an inter-arrival time of 40 seconds except `app1` whose workload was monotonically increased (i.e., inter-arrival time was decreased from 40 to seven seconds). Figure 5.9 presents both before (a) and after (b) snapshots of resource allocations. Note that both `app1` (dark blue) and `app7` (yellow) are resident on the same PM. Initially, `app1` is allocated approximately 300 MIPS while `app7` is allocated approximately 310 MIPS. After the



(a)



(b)

Figure 5.9: A snapshot of resource allocation both (a) before and (b) after an increase in the workload is detected.

increase in the workload, `app1` is allocated approximately 510 MIPS while `app7` is allocated approximately 100 MIPS.

5.5.3 Assessing the Scalability

In addition to the experiments performed in cases studies one and two, the relationship between the number of simulated VMs and the optimization time per step was

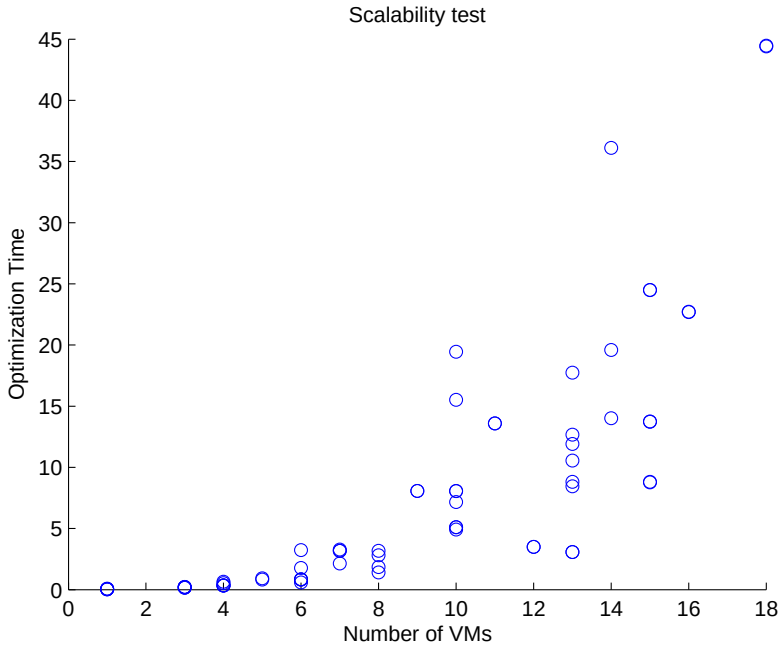


Figure 5.10: The relationship between the number of simulated VMs and the optimization time for each optimization step.

also considered and a non-linear but polynomial relation was observed (see Figure 5.10).

5.6 Summary

The purpose of this chapter was to demonstrate the advantage of "adaptive empirical" models relative to "static empirical" models in resource management. We investigated dynamic tuning of empirical resource allocation models. We described an approach which dynamically updates an empirical model for each application

at runtime in order to capture the effects not considered in the initial specification of the model. This dynamic updating results in more accurate estimates being passed to the optimizer, allowing for better resource utilization on a global scale. We traced the effect of this tuning, in the overall system performance. We showed, in terms of performance, dynamic adjustment of model parameters using an EKF, can outperform a model built off-line using a non-linear regression.

6 Optimal Service Replica Placement via Model Predictive Control

This chapter presents a dynamic service placement algorithm for Optimal Service Placement (OSP) of a set of N-tire software applications. The algorithm solves an optimization problem in each control step. The solution of the optimization suggests that some service replicas be added or removed from the available hosts. These deployment changes are optimal with regards to overall long-term objectives. In addition, the optimization considers restrictions imposed on the number of possible service migrations at each time interval.

6.1 Problem Components

The definition of the OSP problem includes five components: hardware structure, software structure, workload, infrastructure cost, and QoS related Service Level Objectives (SLOs). The solution to the problem is an optimal set of placement changes at each timestep during the lifetime of a cloud provider. Each placement

change is an addition or deletion of a service replica to or from a host in a data center. In the rest of this section, components of the problem are briefly discussed:

6.1.1 Hardware structure

We assume a cloud with heterogeneous resources composed of H hosts. Each host has a number of resources such as disks, random access memory (RAM), and processors. We assume there are K[29] of these types of resources within the data center[30]. The multiplicity of resource k of a host h is denoted by $\Omega_{h,k}$. If the only resource under consideration is the CPU, then the multiplicity can be simply written as Ω_h. Each host resource is also associated with speed factor $\sigma_{h,k}$. If the only resource of consideration is the CPU, then speed factors are written as σ_h.

6.1.2 Software structure

We assume the system is composed of S services. The software structure is modelled based on interactions of these services. We quantify these interactions using two sets (also elaborated in subsection 2.4.1.2): (i) The service demands of the services on different resource types of data center hosts. (ii) The service call graph, where

[29]We use an uppercase letter to denote the number of instances of an entity while using lowercase letters for indexing individual instances.

[30]Note that besides hosts, a data center is also composed of a network and a storage fabric. Although the theory developed here can be tailored for those as well, these resources are not addressed in this chapter.

each edge from a caller service to a callee is labeled by the number of invocations for each execution of the caller.

6.1.3 Workload

The workload component was described in subsection 2.4.1. The workload component it is represented by C classes of users that make use of the shared services. Each class c of users is identified by: (i) it's number of consumers, N_c, and it's think time, Z_c, as discussed in subsection 2.4.1.1, (ii) the number of visits from classes to front-end services derived from CBMG is $V_{c,s}$, denoted for each class c and service s.

6.1.4 Infrastructure cost

The infrastructure cost is based on energy consumption and is modelled as a function of resource utilizations.

6.1.5 Service level objectives

An SLO for each application (or class of users) is represented by an upper bound on the application's response time R_c^{SLA} or lower bound on its throughput X_c^{SLA}. As an example, consider a service center composed of 14 services ($S = 14$), accessed by 2 classes of users ($C = 2$). Figure 6.1 represents the call graph of the services and the

number of visits from the classes to services ($V_{c,s}$); the think times (Z_c) and the number of users (N_c) of each class over time, and the invocation numbers within services ($y_{c,s}$). There are 6 available hosts to support the services. Capacity of the hosts are $\Omega = [16\ 6\ 6\ 16\ 6\ 6]^T$ and their CPU speed factor is $\sigma = [1\ 1.2\ 0.9\ 1.1\ 0.8\ 1.2]$. Hosts are initially unallocated, and no replica is deployed on them.

6.2 Motivating Example

In this section, using a simple example, we show the effect of solving a service placement problem using then optimal control framework. We show that if the objective is to avoid reconfiguration (i.e. there is a cost to service replication changes), and the workload has oscillations, the model predictive control provides better results than stepwise optimization.

Consider the problem of service placement in a small-sized service center where the objective is to minimize the hosting costs and to minimize the total number of relocations while meeting the multi-class workload response time goals. There is a random resource cost coefficient associated with each host. The response time SLA for all the steps were set to $R_t^{\text{SLA}} = [0.146\ 0.267]^T$. The R_t^{SLA} specifies the upper bound on the response time values for classes, namely R^{SLA}.

Assume a three step-ahead prediction of the mean number of users given the observations up to now ($E[N_{c,t+k}|N_{1:t}]$) is provided through some statistical prediction

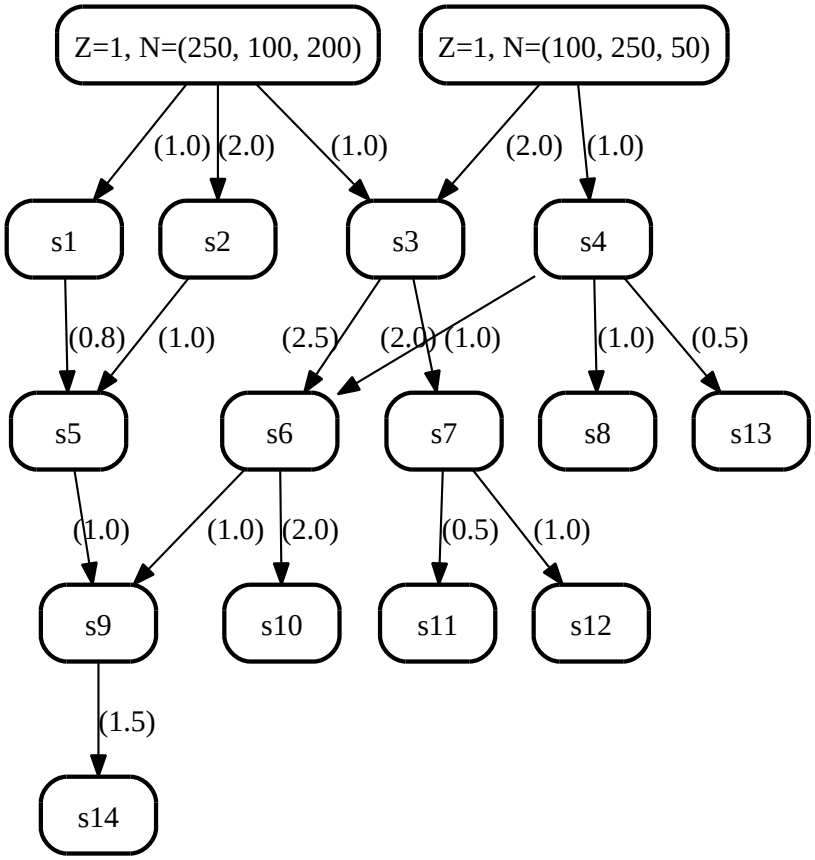


Figure 6.1: An example small scale service center: the call graph of services and classes, the number of calls from classes to services (a portion of y), the invocation numbers within services (another portion of y), the think times (Z) and the number of users (N) of each class over time.

technique:

$$N = \begin{pmatrix} 220.0 & 150.0 & 200.0 \\ 100.0 & 170.0 & 110.0 \end{pmatrix}$$

where each row represents a class of users, and each column represents a future timestep.

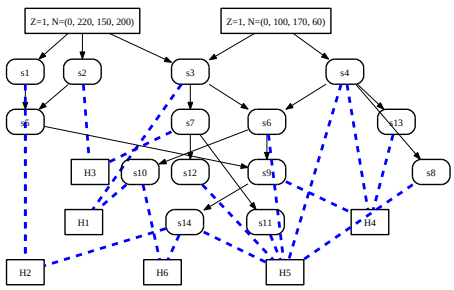
The optimization was applied in two settings: (i) solving a set of individual optimizations for separate steps ignoring the transition cost (step-by-step optimization) and (ii) solving for all the steps in one optimization problem considering the cost of reconfiguration (overall optimization).

In the step-by-step optimization, the optimizer achieves the desired throughput by an initial deployment of 18 replicas in the first step and a total addition of 9 and removal of 11 in the subsequent steps (total changes of 38). The placements are depicted in Figure 6.2. The undirected blue dashed lines denote the placements that have been added in the corresponding steps, and the red dashed lines represent the placements that have been just removed at each time step. The black dashed lines denote the placements performed in the past time steps.

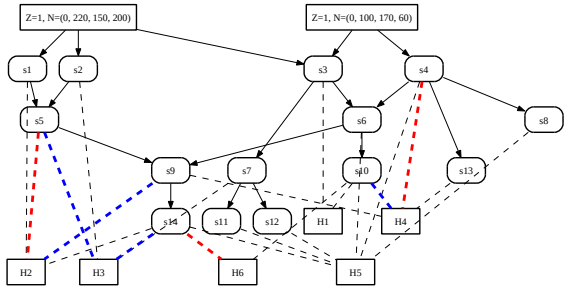
The excessive number of service reallocations in this case are because the optimizer targets the least cost configuration (based on the host coefficients) without taking into account the cost of relocations. The removal and addition of services incur a reconfiguration cost and network overhead, and thus makes this solution

suboptimal.

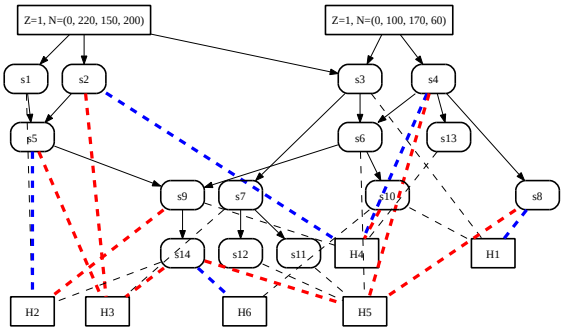
Note that associating cost with the number of relocations in the current step, in case of step-by-step optimization, will not solve the problem. This is because the effect of the relocations propagates through future steps (depending on the future workload).



(a) Initial placement decision (at $t = 1$)



(b) Placement decision at $t = 2$



(c) Placement decision at $t = 3$

Figure 6.2: An example of placement decisions for a small service center using step based optimization, ignoring the future steps.

In the overall optimal case, the desired throughput is achieved by an initial

deployment of 19 service replicas in the first step and the total addition of 0 and the removal of 0 in the subsequent steps (total changes of 19). Although this solution is not as efficient in terms of the resource cost, it has a better total cost since it also considers the relocation cost.

Note that, in this example we assumed a prediction of the future mean workload conditioned on the past values is available through statistical modeling.

6.3 Problem Formulation

Having the above set of inputs, we can formulate the problem as follows: assume a three-dimensional array or tensor θ, whose elements are denoted by $\theta_{s,h,t}$, represents the placement of the replicas of the services on the available hosts over a time horizon. In fact, each $\theta_{s,h,t}$ denotes the portion of service s that is allocated to a host h through a placed replica and a proper routing at a time t. As a result, we have:

$$\sum_{h=1}^{H} \theta_{s,h,t} = 1 \qquad \text{for each } s \text{ and } t \tag{6.1}$$

In other words, at each time step t, θ represents a $H \times S$ bipartite graph of the associations between the hosts and the services. The actual demand on each resource is obtained by taking into account this association of the services and the

hosts ($\theta_{s,h,t}$) as:

$$d_{c,s,k}\theta_{s,h,t} \qquad \text{for each } c,s,h,k,t \tag{6.2}$$

This expression gives the demand of class c on a resource type k of a server h through service s at time t, and $\theta_{s,h,t}$ denotes the portion of service s that is associated to host h through a placed replica and proper routing. Focusing only on the CPU resource, one can drop the index k representing the resource type from $d_{c,s,k}$, for simplicity. For service s, a desired $\theta_{s,h,t}$ at time t can be achieved by dividing the service invocations made to s between its service replicas proportional to $\theta_{s,h,t}$ (taking h as the variable):

$$y_{e,s,h} = \theta_{s,h,t} y_{e,s} \tag{6.3}$$

where $y_{e,s,h}$ denotes the call multiplicity from each service e to a replica of s on h. $\theta_{s,h,t}$ is, in fact, the proportion of the service s requests routed to its replica on the host h at time t. If for some s and h, $\theta_{s,h,t} = 0$, then there will not be any replica of service s on host h at time t. Thus, θ represents both the deployment and the quantity of resource allocation.

The cost to a cloud provider depends on the number of active hosts utilized over the providers' lifetime. The fewer active hosts used, the less cost will be incurred. Indirectly, the cost depends on the placements during the provider lifetime (i.e. $\theta_0, ..., \theta_T$), where the provider lifetime is denoted by T.

The problem is choosing a sequence of optimal placements $(\theta_0^*, ..., \theta_T^*)$ that minimizes the long term resource cost while trying to satisfy the SLOs over time. In summary, the problem is the optimal deployment of the service replicas on the available hardware over time:

$$\begin{aligned}\underset{\theta_0,...,\theta_{T-1}}{\text{minimize}}\ E\Bigg[& r_{\text{SLA}} \sum_{t=1}^{T} \sum_{c=1}^{C} \lambda_{\text{SLA}}(R_{c,t} - R_c^{\text{SLA}}) \\ & + r_{\text{resource}} \sum_{t=1}^{T} \sum_{h=1}^{H} \lambda_{\text{resource},h}(U_{h,t}) \\ & + r_{\text{dep}} \sum_{t=1}^{T} \sum_{h=1}^{H} \sum_{s=1}^{S} \lambda_{\text{dep}}(\theta_{s,h,t}, \theta_{s,h,t-1}) \Bigg]\end{aligned}$$

subject to:

$$\begin{aligned} R_t &= \text{LQM}_{\Omega,\sigma}(d_t, \theta_t, W_t) && \forall t \\ \sum_{h=1}^{H} \theta_{s,h,t} &= 1 && \forall s, t \end{aligned}$$

Here, T denotes the service centre lifetime. $(u_0, ..., u_{T-1})$ is the service placements performed during this lifetime. $E[...]$ represents an expected value. r_{SLA}, r_{resource}, and r_{dep} respectively represent the coefficients associated with the cost of violating SLAs, the cost of hardware resources, and the cost of changes in the deployment of services. These coefficients (r_{SLA}, r_{resource}, and r_{dep}) are used to tune the trade-off between the different cost factors. The expression $\lambda_{\text{SLA}}(R_{c,t} - R_c^{\text{SLA}})$ represents the cost of violating the SLO for a class c at time t. Note that the SLO is defined only based on the response time of the class c at each timestep (i.e., $R_{c,t}$).

$\lambda_{\text{resource},h}(U_{h,t})$ represents the cost incurred by a host h, where the cost of a resource is calculated based on its utilization. The expression $\lambda_{\text{dep}}(\theta_{s,h,t}, \theta_{s,h,t-1})$ represents the cost of modifications in the deployment, and is calculated based on the modifications at each time step t. Here, $R_t = [R_{c,t}, ..., R_{c,t}]^T$ denotes the response times of the user classes over time, d_t denotes the set of service demands of the classes on the services (i.e. the matrix $(d_{c,s,t})$) for each t, W_t is the non-stationary stochastic workload of classes (i.e. $W_t = [W_{1,t}, ..., W_{c,t}]^T$). The three cost functions (λ_{SLA}, $\lambda_{\text{resource},h}$, and λ_{dep}) are described in detail in the subsection 6.3.2.

6.3.1 Equality Constraints

The first equality constraint is an output equation which models the QoS attributes. Here, the QoS attributes are the response times for the classes of users (i.e. R_t). R_t is the result of the current deployment θ_t, the workload W_t, the software structure (d_t) and the structure of the data center (Ω and σ).

Note, here we assume that the control interval is large enough that system stabilizes at each step, and, as a result, we can ignore the transient queuing dynamics and use the MVA of queuing networks to derive the output equation. Thus, the workload values are directly fed into the output equation.

6.3.2 Cost elements

In the optimization problem, to model the SLA violation cost, the function λ_{SLA} is directly applied to the response time deviation of each class. The definition of λ_{SLA} is as follows:

$$\lambda_{\text{SLA}}(x) = max\{x, 0\} \tag{6.4}$$

and it is a convex and increasing function. Applying this function to $(R_{c,t} - R_c^{\text{SLA}})$ means we desire to impose a cost whenever the response time for a class exceeds its target SLA. In other situations where other behaviours are expected, λ_{SLA} can be redefined using other convex functions.

The total cost of deployment in a data center is comprised of the cost of electricity consumed by its physical machines. The function $\lambda_{\text{resource}}(U_{h,t})$ is derived from the model of electricity consumption of the host. It is defined as:

$$\lambda_{\text{resource},h}(U_{h,t}) = \begin{cases} \sigma_h U_{h,t} + \sigma'_h & \text{for } U_{h,t} \in (0, \Omega_h] \\ 0 & \text{for } U_{h,t} = 0 \end{cases} \tag{6.5}$$

where σ_h and σ'_h are host specific constants. An active server with a minimum load consumes roughly 50% of the electricity of a fully loaded server. In addition, if we model the standby servers, with an absolute zero load, the resulting cost function will have a discontinuity at the 0 load ($U = 0$). This suggests that a minimum cost associated with a data center can be achieved by consolidating all the workloads in

a minimum number of active machines.

The function $\lambda_{\text{dep}}(x, y)$ represents the cost of deployment changes. It is defined as the sum of the additions or removal of the service replicas to the hosts:

$$\lambda_{\text{dep}}(\theta_{s,h,t-1}, \theta_{s,h,t}) = \begin{cases} 1, & (\theta_{s,h,t} \neq 0 \oplus \theta_{s,h,t-1} \neq 0) \\ 0, & \text{otherwise} \end{cases} \tag{6.6}$$

where $\oplus$ denotes the logical Xor. $\lambda_{\text{dep}}(\theta_{s,h,t-1}, \theta_{s,h,t})$ is 1 in two cases: (i) if the service s was deployed at time $t-1$ on the host h and is not deployed there at t, (ii) if the service s was not deployed at time $t-1$ on the host h and is deployed there at t.

6.4 A Fast Solution through MPC

There are several subtleties with the form of the proposed problem. In the following subsections, we discuss each and provide a solution.

6.4.1 Dealing with the Non-linearity of the LQM

The first subtlety in solving the described allocation problem is that the output equation of the system dynamics uses a nonlinear function, namely the LQM. Modelling the response times of the user classes needs a non-linear set of equalities. The LQM is an iterative algorithm for which there is no closed form general solution

available.

To address the non-linearity of the LQM, instead of directly using the LQM, we use its derived performance bounds. First, consider the following inequality from Queuing Networks Models (QNM):

$$U_{h,k} < \Omega_{h,k} \ \forall h, k \tag{6.7}$$

The inequality 6.7 states that the capacity of a resource can be used as an upper bound to its utilization[31]. Inequality 6.7 is an implication of the queuing theory and holds regardless of the type of workload to be applied to the system (open or closed).

Also, consider the following equation, derived from queuing theory, which relates utilization of resources to the throughput of user classes:

$$U_{h,k,t} = \ \sum_{c=1}^{C} \left(X_{c,t} \sum_{s=1}^{S} (d_{c,s,k} \theta_{s,h,t}) \right) \quad \forall h, k, t \tag{6.8}$$

This equation is derived from the equations 8.2 and 6.2.

Using equation 6.8, we rewrite the inequality 6.7 regarding the physical constraints as:

$$U_{h,k,t} = \sum_{c=1}^{C} \sum_{s=1}^{S} X_{c,t} d_{c,s,k} \theta_{s,h,t} < \Omega_{h,k} \ \forall h, k, t \tag{6.9}$$

[31]Note that here we used the utilization in a slightly different way than in queuing theory. In fact, to obtain the utilization in the queuing theory sense, what we refer to as utilization here should be divided by the resource multiplicity. In the original queuing theory formula $U \leq 1$.

Inequality 6.9 is concerned with the heavy load (or saturation) situation. It states "each hardware component limits the maximum possible throughput that the system can achieve. Since the bottleneck center is the first to saturate, it restricts the system's throughput most severely"[55].

It is important to note that in case of a lightly loaded (non-saturated) system, there is another tighter upper-bound for throughput. "The largest possible throughput for a lightly loaded system occurs when each additional request is not delayed at all by any other requests in the system. In this case for each class c, no time is spent queuing, $d_c = \sum_{h=1}^{H} \sum_{k=1}^{K} \sum_{s=1}^{S} d_{c,s,k}\theta_{s,h,t}$ time units are spent in service, and Z_c time units are spent thinking, so the each class throughput is $N_c/(Z_c + d_c)$"[55]. The upper bound can thus be derived from the following equation:

$$X_{c,t} \leq N_{c,t}^{\text{pred}} / \left(\sum_{h=1}^{H} \sum_{k=1}^{K} \sum_{s=1}^{S} d_{c,s,k}\theta_{s,h,t} + Z_c \right) \tag{6.10}$$

This throughput upper bound, however, is not our concern, because with the assumption that the response time SLAs are valid (i.e. more than the sum of service demands), the throughput SLAs will be naturally less than this upper-bound and the controller always tries to only meet this SLA; it would not waste resources and exceed the SLA when it does not reduce the cost.

We also need to describe the response time SLOs (R_c^{SLA}) in terms of throughput constraints. This is done based on the assumption of finite user populations. If $N_{c,t}^{\text{pred}}$ and $Z_{c,t}^{\text{pred}}$ can be predicted and target response time R_c^{SLA} is given, then the

constraint based on the response time (i.e. $R_{c,t} < R_c^{\text{SLA}}$) can be translated to a constraint in terms of the throughput as:

$$X_{c,t}^{\text{SLA}} = N_{c,t}^{\text{pred}} / (Z_{c,t}^{\text{pred}} + R_{c,t}^{\text{SLA}}) \qquad \forall c, t \qquad (6.11)$$

$$X_{c,t} > X_{c,t}^{\text{SLA}} \qquad \forall c, t \qquad (6.12)$$

where $N_{c,t}^{\text{pred}}$ denotes the predicted mean number of users of a class c at a future time t and $Z_{c,t}^{\text{pred}}$ denotes the predicted mean think time of the users of a class c at a future time t. Note that, the equation 6.11 can be calculated in advance, and the resulting X_c^{SLA} can be used in several optimizations over time.

6.4.1.1 Approximating the Effects of Resource Contention

In order to derive a fast solution, we neglect the effect of the resource contention resulting from the stochastic think times and the service demands. In the presence of random think times (in a case of a closed workload) and random arrival rates (in the case of an open workload) queues build up at resources. We refer to this queue build-up as the hardware contention. This contention reduces the classes throughputs, increases the response times and might even shift the bottleneck from one resource to another. Instead of evaluating the effect of contention, we try to move the system into a region where the effect of the contention is minimized. Usually contention happens where resources are highly utilized. Keeping the system

in a region where its resource utilizations are controlled (usually between %58 to %80 of the capacity Ω_h) is a common way to avoid contention[32] [12]. Thus, the inequality 6.7 might appear in the final optimization as:

$$U_{h,k} \leq L\Omega_{h,k} \ \forall h, k \tag{6.13}$$

where L is a constant usually set to the value between 0.6 and 0.8.

6.4.2 Non-linearity in the Resource Cost

The function $\lambda_{\text{resource}}$ introduced in the equation 6.5 is nonlinear because it has a discontinuity at $U = 0$ (i.e. when a physical machine is in standby mode). We approximate this function, based on the utilization law of the equation 6.8 as:

$$f(h) = \sum_{s=1}^{S} \sigma^*_{s,h} \sum_{c=1}^{C} X_{c,t} d_{c,s} \theta_{s,h,t}$$

Where each $\sigma^*_{s,h}$ is a cost coefficient. This cost coefficient can be based on the original values of σ_h and σ'_h. For example [59] derives similar coefficients through linear approximation of the original resource cost function. The coefficients σ_h and σ'_h can be also based on some heuristics for example system administrator's preferences. For example, for our cases studies, we used the following formula to derive $\sigma^*_{s,h}$: $\sigma^*_{s,h} = rand_{H,1} * \mathbf{1}_{1,S} + 10$. In this formula, $rand_{H,1}$ are sampled from a random variable with the uniform distribution, $U(0, 1)$.

[32] This is sometimes referred to as the headroom principal [47].

Also note that, the degree of the sparsity of the solution, and sometimes the convergence of the optimization depends on the way the elements of $\sigma^*_{s,h}$ are generated. We realized that to guarantee the sparsity in our solution it is mandatory to have different cost coefficients for different physical machines.

6.4.3 Nonlinearity in the Reconfiguration Cost

Let us assume service placements $(\theta_0, ..., \theta_T)$ are outcomes of a process driven by a set of placement actions $(u_0, ..., u_T)$. We assume that θ_t is simply a discrete-time integral (or accumulation) over the placement actions:

$$\theta_t = \theta_{t-1} + u_t \tag{6.14}$$

Equation 6.14 represents the dynamics of the placement or the state transition. Note that θ_t is solely driven by the placement decisions $(\{u_0, ..., u_{T-1}\})$ deterministically (i.e., service center is fully predictable). The constraints resemble an ordinary differential equation with θ_t as a state. We redefine the reconfiguration cost of equation 6.6, over the placement decisions $(\{u_0, ..., u_{T-1}\})$ using the following stage cost function:

$$\lambda_{\text{dep}}(\theta_{s,h,t-1}, \theta_{s,h,t}) = |u_{s,h,t}| \tag{6.15}$$

6.4.4 Considering the Stochastic Workload

Finally, to solve the abstract stochastic planning problem of Section 6.3, we converted it into several step-by-step deterministic optimizations over the cloud lifetime. Each optimization takes place on arrival at each new step by taking the current state as an initial point of planning. Further, in each optimization, the planning is only done for a limited number of future steps, which is referred to as the *lookahead horizon* and denoted by J; thus reducing the number of steps involved in the optimization from $T-t$ to J (i.e., this assumes that $J << T-t$). The stochastic workload is also considered. At each time-step t, workloads within the lookahead horizon $\{W_j\}_{t\leq j\leq J}$ are fixed at their predicted conditional mean value (i.e., $E[W_j|W_{1:t}]$) based on a Dynamic Linear Model (DLM)[110]. Then, at each step, a perfect information[33], deterministic optimal control problem with a limited horizon is solved once, and the control value to be applied is derived [34]; everything is repeated once a new observation is available. The deterministic optimal control program, solved at each step, is represented in Figure 6.3. We describe further details of this program below.

In this mathematical program: R_c^{SLA} denotes the response time SLO for each class c in terms of the upper bound. J denotes the provider's lifetime which here is

[33]In a perfect information control problem the value of all the state variables are known (either observed or have been estimated).

[34]In original MPC this value is the first value of the planning sequence.

$$
\begin{aligned}
\text{given: } & R^{\text{SLA}}_{c,j}\ \forall c \forall j, \theta_{s,h,0}\ \forall s, \forall h, \Omega_h\ \forall h, \\
& t, J, r_{\text{resource}}, r_{\text{dep}}, r_{\text{SLA}} \\
\underset{u_0,\ldots,u_{t-1}}{\text{minimize}}\ & r_{\text{SLA}} \sum_{j=t}^{J} \sum_{c=1}^{C} pos(X^{\text{SLA}}_{c,j} - X_{c,j}) \\
& + r_{\text{resource}} \sum_{j=t}^{J} \sum_{h=1}^{H} \sum_{s=1}^{S} \sigma^{*}_{s,h} \sum_{c=1}^{C} X_{c,j} d_{c,s} \theta_{s,h,j} \\
& + r_{\text{dep}} \sum_{j=t}^{J} \sum_{h=1}^{H} \sum_{s=1}^{S} |u_{s,h,j}| \\
\text{subject to: } & X^{\text{SLA}}_{c,j} = N^{\text{pred}}_{c,j} / (Z^{\text{pred}}_{c,j} + R^{\text{SLA}}_{c,j}) && \text{for each } c,j \\
& \theta_{s,h,j+1} = \theta_{s,h,j} + u_{s,h,t} \\
& \sum_{h=1}^{H} \theta_{s,h,j} = 1 && \text{for each } s,j \\
& U_{h,j} \leq L \Omega_h \sigma_h && \text{for each } h,j \\
& U_{h,j} = \sum_{c=1}^{C} X_{c,j} \sum_{s=1}^{S} d_{c,s} \theta_{s,h,j} && \text{for each } h,j
\end{aligned}
$$

Figure 6.3: The deterministic optimal control program, solved at each step, providing an answer to the abstract stochastic planning problem of Section 6.3.

our control window size. Ω_h denotes the multiplicity of a host h. r_{resource} denotes the infrastructure cost coefficient. r_{dep} denotes the coefficient associated with moving

the service replicas. $\{u_0, ..., u_{t-1}\}$ represents the sequence of placement changes for the service replicas. $E[...]$ represents the expected value of a random variable. H represents the number of hosts. S represents the number of services. $\sigma^*_{s,h}$ represents the artificial cost coefficients for each service-host pair (i.e. each replica placement). C represents the number of classes. $d_{c,s}$ represents the demand of a class c on a service s. Note that in this program we only focused on one resource type, CPU. Thus, we dropped the index k from the original demands $d_{c,s,k}$. $\theta_{s,h,t}$ represents the distribution of the service replicas over the classes. A matrix composed of elements $(\theta_{s,h,0})$ represents the initial deployment of the service replicas on the hosts. The symbol $|.|$ represents an absolute value of a scaler. $X_{c,t}$ represents the throughput of a class c at time t. $X^{\text{SLA}}_{c,t}$ represents the throughput SLO of a class c at a time t. ($N^{\text{pred}}_{c,t}$ denotes the predicted mean number of users of a class c at a future time t. We are going to elaborate on the nature of this prediction in the subsection 6.4.4.) $Z^{\text{pred}}_{c,t}$ denotes the predicted mean think time of the users of a class c at a future time t. R^{SLA}_c represents the response time of a class c. $u_{s,h,t}$ represents a placement change for a service s on a host h at a time t. $N_{c,t}$ represents the population of a class c at a time t. $U_{h,t}$ is the utilization of the host h at the time t.

The solution of this optimization problem gives the optimal value for $\{u_{s,h,t}\}$ and $\{\theta_{s,h,t}\}$. Essentially the above optimization problem makes the controller follow these principles: (i) to minimize the SLA violation cost, the controller has to follow

the workload when it increases, (ii) in order to minimize the infrastructure cost the controller has to follow the workload when it decreases, (iii) in order to minimize the reconfiguration cost, the controller has to stop following the workload once the workload is too variable or deviates too much.

6.4.5 A Solver Friendly Format

Looking at the last line of the program 6.3, one can note that the equality constraint, which originates from the equality 6.8, is not in a *disciplined convex form* [41]. This is because X_c and $\theta_{s,h}$ are free variables which are not combined in a linear or quadratic form. Thus, it cannot be implemented in some convex optimization solvers as is. However, since the θ's are proportions (see equation 6.2) we can write the equation in a disciplined convex form. Assuming that we are only dealing with one type of resource (i.e. CPU), and ignoring the time aspect for simplicity, let us define the following variables:

$$\mu^{\gamma}_{c,s} = X_c d_{c,s}$$

$$\mu^{\theta}_{s,h} = \sum_{c} X_c d_{c,s} \theta_{s,h}$$

These variables have the same units as the utilization. Having these variables, we can say:

$$\begin{aligned}\sum_h \mu^{\theta}_{s,h} &= \sum_h \sum_c X_c d_{c,s} \theta_{s,h} \\ &= \sum_h \sum_c \mu^{\gamma}_{c,s} \theta_{s,h} \\ &= \sum_c \mu^{\gamma}_{c,s} \sum_h \theta_{s,h} \\ &= \sum_c \mu^{\gamma}_{c,s} \end{aligned} \tag{6.16}$$

and, according to equation 6.8 we have:

$$\begin{aligned}U_{h,k} &= \sum_c X_c \sum_s (d_{c,s,k} \theta_{s,h}) \\ &= \sum_s \sum_c X_c d_{c,s} \theta_{s,h} \\ &= \sum_s \mu^{\theta}_{s,h} \end{aligned} \tag{6.17}$$

Using equations 6.16 and 6.17, the inequality constraint 6.9 can be presented in a solver friendly format as:

$$\sum_c \mu^{\gamma}_{c,s} = \sum_t \mu^{\theta}_{s,h} \qquad \text{for each service } s = 1...S \tag{6.18}$$

$$\sum_s \mu^{\theta}_{s,h} < \Omega_h \sigma_h \qquad \text{for each host } h = 1...H \tag{6.19}$$

One interpretation of the above formulas is as follows: $\mu^{\theta}_{s,h}$ is the portion of a host h's resource (quantified in a standard way using σ_h) that is associated to service s.

Note that the value of $\theta_{s,h}$ can be recovered as follows:

$$\begin{aligned}\theta_{s,h} &= \frac{\sum_c X_c d_{c,s} \theta_{s,h}}{\sum_c X_c d_{c,s}} \\ &= \frac{\sum_c X_c d_{c,s} \theta_{s,h}}{\sum_h \sum_c X_c d_{c,s} \theta_{s,h}} \\ &= \frac{\mu^{\beta}_{s,h}}{\sum_h \mu^{\beta}_{s,h}}\end{aligned} \tag{6.20}$$

and X_c can be recovered as:

$$X_c = \frac{\mu^{\gamma}_{c,s}}{d_{c,s}} \tag{6.21}$$

The solver friendly version of the optimization of figure 6.3, derived using the above method is presented in Figure 6.4.

The solution of the program of Figure 6.4 gives the optimal value for $\mu^{\theta}_{s,h,t}$ and $\mu^{\gamma}_{c,s,t}$ respectively denoted as $\mu^{\theta*}_{s,h,t}$ and $\mu^{\gamma*}_{c,s,t}$. From these, one can recover $\theta^{*}_{s,h,t}$ using equations 6.20 and 6.21.

6.5 Placement and Considering the Effect of the Contention

The assumption of deterministic workloads will result in an underestimation of response times and overestimation of throughputs. This is because random arrivals generate queues, which in turn decreases the throughputs and increases the response times. Depending on the distribution of the arrival rates or think times and the demands, the system queues will be stabilized in different queue lengths. Thus

$$
\begin{aligned}
\text{given: } & X^{\text{SLA}}_{c,j} && \text{for each } c, j, \\
& \mu^{\theta}_{s,h,0} && \text{for each } s \text{ and } h, \\
& \Omega_h && \text{for each } h, \\
& t, J, r_{\text{resource}}, r_{\text{dep}}, r_{\text{SLA}} && \\
\underset{u_0,\ldots,u_{t-1}}{\text{minimize}} \; & r_{\text{SLA}} \sum_{j=t}^{J} \sum_{c=1}^{C} pos(X^{\text{SLA}}_{c,j} - X_{c,j}) && \\
& r_{\text{resource}} \sum_{j=t}^{J} \sum_{h=1}^{H} \sum_{s=1}^{S} \sigma^{*}_{s,h} \mu^{\theta}_{s,h,j} && \\
& + r_{\text{dep}} \sum_{j=t}^{J} \sum_{h=1}^{H} \sum_{s=1}^{S} |u_{s,h,j}| && \\
\text{subject to:} & && \\
& \mu^{\theta}_{s,h,j+1} = \mu^{\theta}_{s,h,j} + u_{s,h,t} && \\
& U_{h,j} \leq L \Omega_h \sigma_h && \text{for each } h,j \\
& U_{h,j} = \sum_{s} \mu^{\theta}_{s,h,j} && \text{for each } h,j \\
& \sum_{h} \mu^{\theta}_{s,h,t} = \sum_{c} \mu^{\gamma}_{c,s,t} && \text{for each } s,\ j \\
& \mu^{\gamma}_{c,s,t} = d_{c,s} X_{c,t} && \text{for each } c \\
& \mu^{\theta}_{s,h,t} \geq 0, \mu^{\gamma}_{c,s,t} \geq 0 && \text{for all } h,\ j,\ s,\ k
\end{aligned}
$$

Figure 6.4: The solver friendly version of the optimization of Figure 6.3.

there are situations where the constraint $R_c \leq R_c^{\text{SLA}}$ is satisfied but, the response time of the class c requests considering the contention (denoted by R_c^Q) is not less than the specified SLA (i.e. $R_c^Q \leq R_c^{\text{SLA}}$ does not hold).

To offset this underestimation, the constraint $X_c \geq X_c^{\text{SLA}}$ is substituted with a tighter bound: $X_c \geq X_c^{\text{target}}$ than the SLA one. This bound represents a reservation for the processing capacity needed to support SLAs, despite stochastic arrival. The stochastic model is used to compute the throughputs in presence of contention in the system, given a placement θ. One needs to set the target high enough that the solution through the deterministic model satisfies the stochastic SLA target. The iterative Algorithm 3 finds the optimal placement sequence considering the contention in the system using the above technique. It is an adaptation of the algorithm proposed in [60] modified to work in the context of MPC. It iteratively switches between the optimization of the deterministic model and solving the stochastic one.

$X_c^Q(\theta^i)$ in the algorithm, denotes the throughput solution of the queuing model for a class c when the placement is θ^i, obtained from LQN solution of chapter 8 or HQN solution through equation 2.12 to 2.15.

Algorithm 3: The MPC placement considering the effect of contention.

input : A^N, A^Z, N^{est}, Z^{est}, T, R_c^{SLA}, trade-off coefficients $r_{\text{resource}}, r_{\text{dep}}$
output : optimal placement sequence $\theta_{s,h,t}$

1 solve the following linear constraint satisfaction problem to obtain the predicted sequence of workload:

$$N_{t+1}^{\text{pred}} \leftarrow A^N \; N_t^{\text{pred}} \qquad \text{for each } t$$
$$Z_{t+1}^{\text{pred}} \leftarrow A^Z \; Z_t^{\text{pred}} \qquad \text{for each } t$$
$$Z_0^{\text{pred}} \leftarrow Z^{\text{est}}$$
$$N_0^{\text{pred}} \leftarrow N^{\text{est}}$$

2 calculate the sequence of throughput objective based on the predicted workload and desired SLA:

$$X_{c,t}^{\text{SLA}} \leftarrow N_{c,t}^{\text{pred}}./(Z_{c,t}^{\text{pred}} + R_c^{\text{SLA}}) \text{for each } c,t$$

3 initialize $X_c^{\text{target},0} \leftarrow X_c^{\text{SLA}}$
4 **repeat**
5 solve the following optimization problem:

$$\text{minimize } r_{\text{resource}} \sum_t \sum_h \sigma_h U_{h,t} + r_{\text{dep}} \sum_t \sum_h \sum_s |u_{s,h,t}|$$

subject to:

$$\theta_{s,h,t+1} = \theta_{s,h,t} + u_{s,h,t} \qquad \text{for each } c,t$$
$$\sum_h \theta_{s,h,t}^i = 1 \qquad \text{for each } s$$
$$X_c^i > X_c^{\text{target},i} \qquad \text{for each } c$$
$$U_h^i < \Omega_h \qquad \text{for each } h$$
$$U_h^i = \sum_c X_{c,t}^i \sum_s d_{c,s} \theta_{s,h,t}$$

6 $X_c^{\text{target},i+1} \leftarrow X_c^{\text{target},i} + (X_c^{SLA,i} - X_c^Q(\theta^i))$
7 **until** $|X_c^{target,i+1} - X_c^{target,i}| < |0.01 X_c^{target,i}|$;

6.6 Case Study: The Controller Response to a Realistic Workload

In this case study, we assess the reaction of the controller to a stochastic workload taken from a real application. The main objective is to test the behaviour of the controller in terms of its ability to navigate the trade-off among SLA violations, resource cost, and the thrashing cost. The structure of the services, the call graph, and the classes are the same as ones used in the motivating example (section 6.2, Figure 6.1). There are 14 services deployed on 6 hosts, possibly giving 84 service replicas. There are two classes of service utilizing the services.

The workload for classes is a portion of the well-known FIFA98's workload obtained by processing the Web server logs. We processed the workload of the day 42 of FIFA98 to obtain the number of users for each five-minute interval. This gave 288 samples for the whole day (let us denote these by N_t). We then constructed two 12 hour workloads for two class of users based on this as: $N_{c1,t} = N_t - 200$ for the first 12 hours and $N_{c2,t} = \lfloor N_{t+12h}/2 \rfloor - 110$ for the second 12 hours. These values are chosen so that the workloads of classes have peaks and valleys to test the controller. The think times for the classes are $Z = [1, 1]^T$. Multiplicity of hosts are $\Omega = [9, 10, 10, 9, 10, 7]^T$. The speed factors for the hosts are $= [1, 1.2, 0.9, 1.1, 0.8, 1.2]^T$ The service demand for the services are $d = [5, 4, 2, 8, 1, 2, 5, 8, 6, 8, 4, 5, 6, 5] \times 10^{-3}$.

The random cost coefficients are $\sigma^*_{s,h} = rand_{H,1} * \mathbf{1}_{1,S} + 10$. Each variable $rand_h$ is a value sampled from random variable of the uniform distribution, $U(0, 1)$.

The system was simulated using the SimPy [74] simulator. The optimization problem was described using the `cvx` framework [41] in Matlab where the problem is delegated to the SeDuMi [90] solver.

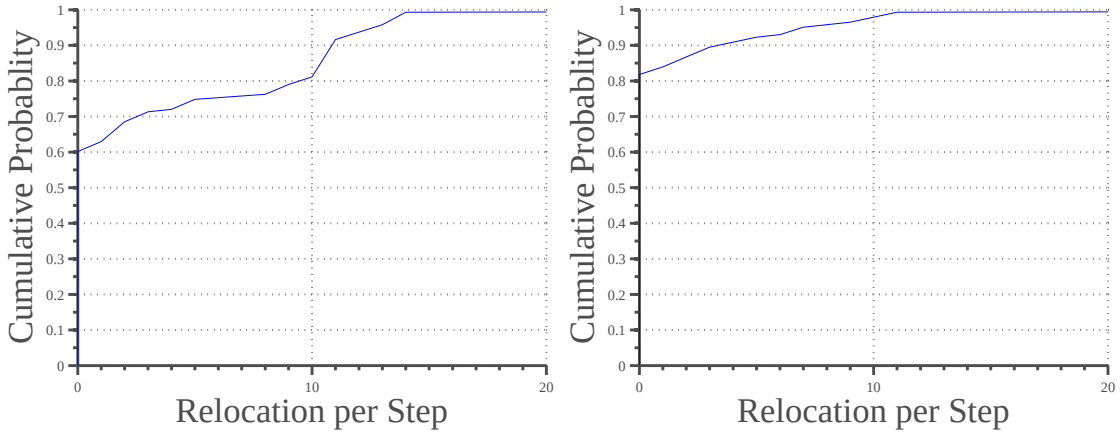
In this case study we performed two sets of experiments. In the first set, we investigated the effect of different cost coefficients on the behaviour of the controller. In the second set of experiments, we investigate the effect of changing the length of lookahead horizon on the achievable cost trade-offs.

6.6.1 First Set of Experiments: Different Cost Coefficients

In the first set of experiments we qualitatively show the ability of the controller to achieve different trade-offs between number of relocations and the number of SLA violations. Different trade-off points will be achieved by changing the cost coefficients.

We performed several experiments where we tweaked the cost coefficients and simulated the controller and the system. Here we show the detailed results for two simulations as representatives. The lookahead horizon for MPC controller is five steps ($J = 5$) for both experiments. In the first simulation, we set the cost coefficients as follows: $[r_{\text{SLA}} = 50, r_{\text{resource}} = 10, r_{\text{dep}} = 4]$. In the second set,

we set cost coefficients to: $[r_{\text{SLA}} = 50, r_{\text{resource}} = 1, r_{\text{dep}} = 40]$. Thus, intuitively the controller in the first configuration cares more about the cost of resources, while in the second configuration it cares more about the cost of reconfiguration. Let us call these controller configurations resource-precedent and reconfiguration-precedent controllers. The result of simulations is depicted in Figures 6.5 to 6.8.

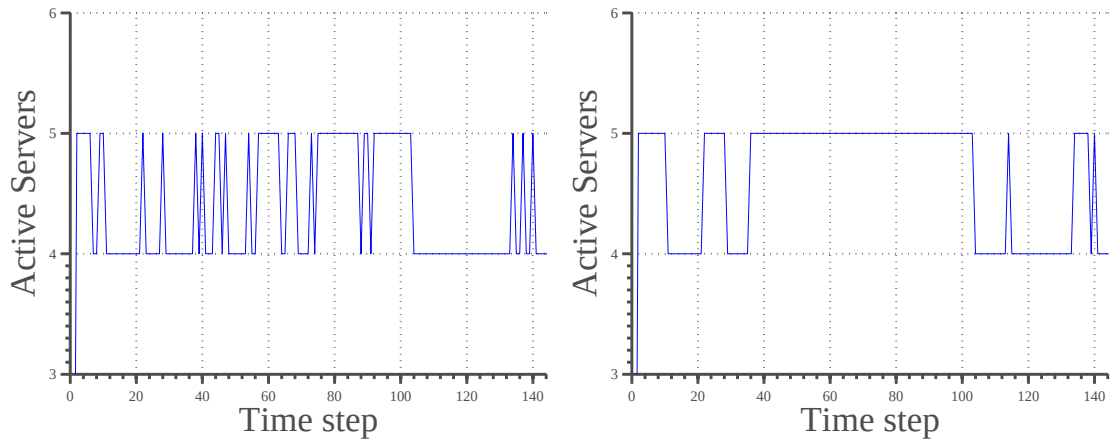


(a) Resource Precedent Controller (b) Reconfiguration Precedent Controller

Figure 6.5: The cumulative distribution function (CDF) of the number of relocations for two different MPC configurations.

Figure 6.5 represents the cumulative distribution function (CDF) of the number of relocations done for each time step for the resource-precedent controller (6.5a) and the reconfiguration-precedent (6.5b). As can be noted, in Figure 6.5a the Y value of the resource-precedent graph (blue solid line) at 0 jumps to 0.6. This means 60% of the area under the PDF curve is placed at 0. In other words, in 60%

of steps, there is no relocation. Compared to this, the CDF of the reconfiguration-precedent controller (blue dashed line) shows a lot fewer relocations per step; it has no relocation in almost 82% of the steps.

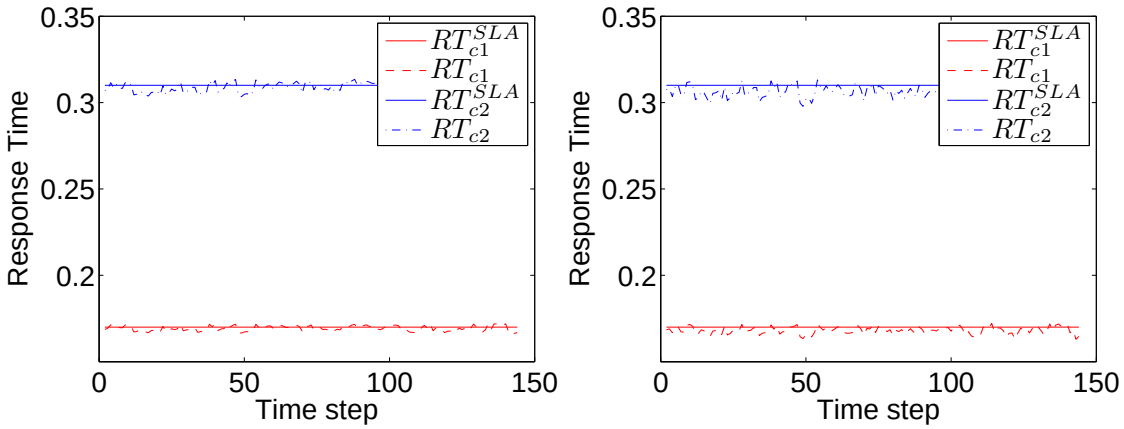


(a) Resource Precedent Controller (b) Reconfiguration Precedent Controller

Figure 6.6: The number of active servers over time for two different MPC configurations.

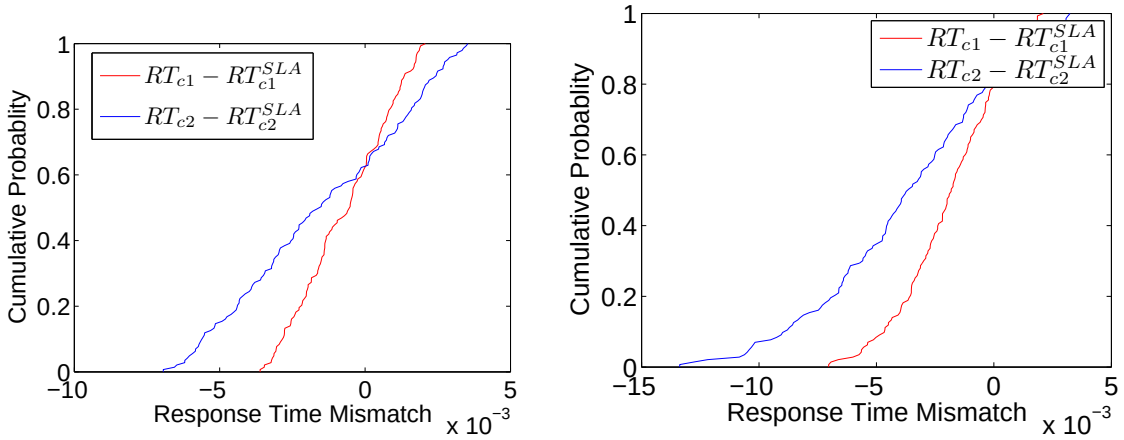
Figure 6.6 represents the number of active servers over time in both configurations. The figure is consistent with our assumption: the resource-precedent controller (6.6a) tries to save resources by deactivating/activating physical machines more frequently than the reconfiguration-precedent one (6.6b).

To show the extent to which the SLA is satisfied in each scenario, we have provided the raw response time graphs in Figures 6.7a and 6.7b. However, it is not very easy to see their difference. In Figures 6.8a and 6.8b we represent the difference



(a) Resource Precedent Controller (b) Reconfiguration Precedent Controller

Figure 6.7: The raw response time graphs over time for two different MPC configurations.



(a) Resource Precedent Controller (b) Reconfiguration Precedent Controller

Figure 6.8: The difference between response time and the response time SLA as a CDF for two different MPC configurations.

between the response time and the response time SLA (i.e. $RT_c - RT_c^{\text{SLA}}$) as a CDF, taking each time step as one sample. In 6.8a, there were some points where the SLA is breached, and that is why the CDF increases into the region $RT_c > RT_c^{\text{SLA}}$. In 6.8b, the response time did not breach the SLA, and the CDF reaches 1 before the region where $RT_c > RT_c^{\text{SLA}}$.

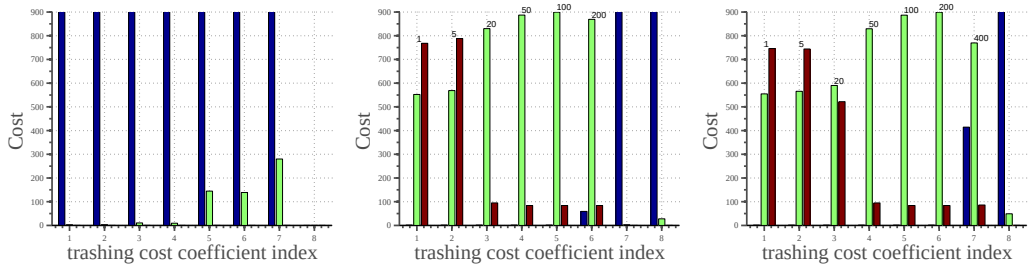
6.6.2 Second Set of Experiments: Different Lookahead Horizons

In this set of experiments we investigate the effect of length of lookahead horizon on the properties of the controller. We also justify our hypothesis that controller with a longer lookahead horizon can achieve more trade-off points in terms of resource, SLA violation, and reconfiguration costs.

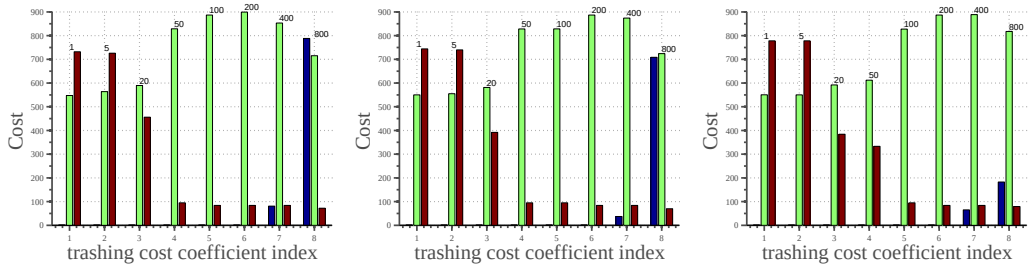
In order to assess the impact of lookahead horizon on the cost elements, we performed an extensive set of experiments. In each of the experiments, we fixed the SLA and resource cost coefficients at a constant value ($r_{\text{SLA}} = 50, r_{\text{resource}} = 1$) and chose the reconfiguration cost coefficient from the following set of values:

$$r_{\text{dep}} \in \{1, 5, 20, 50, 100, 200, 400, 800, 1600, 3200\}$$

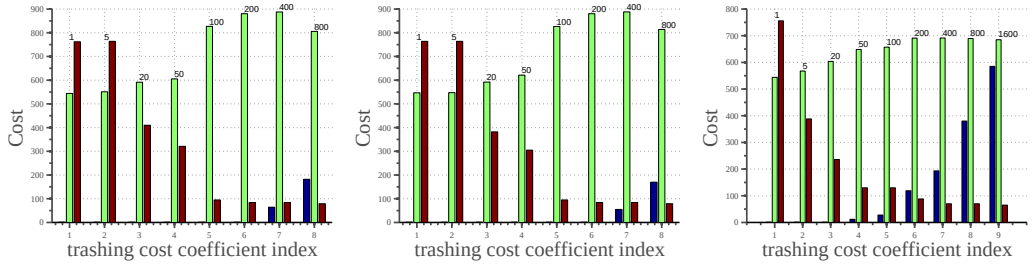
Experiments also iterated between different lookahead horizons from $T = 1$ to $T = 10$. As a showcase, in Figure 6.9 we represent the graphs for eight different lookahead horizons, $T = 1$ (6.9a) to $T = 7$ (6.9g), and one for the optimal controller (6.9i). Costs for the optimal controller are computed the same way as the MPC.



(a) two-step lookahead (T=1) (b) two-step lookahead (T=2) (c) three-step lookahead (T=3)



(d) two-step lookahead (T=4) (e) two-step lookahead (T=5) (f) two-step lookahead (T=6)



(g) seven-step lookahead (T=7) (h) two-step lookahead (T=8) (i) Optimal Controller

Figure 6.9: The cost trade-off curves ((a) to (b)) achieved by the MPC based controllers with the lookahead horizons of $T = 1$ to $T = 8$, and ((i)) achieved by an optimal controller.

Each group of horizontal bars represents one of the values assigned to r_{dep} (see the annotation on top of the middle bar in each group). In each group, the cost of SLA violation (left blue bar) is calculated according to the equation 6.4. The cost of infrastructure (middle green bar) is also calculated based on the original formula 6.5, taking into account the discontinuous resource cost function (as a function of utilization). Finally, the cost of reconfiguration (right red bar) is based on the actual number of service additions/deletions at each interval based on the equation 6.6.

After performing several tests, we observed that with low lookahead horizons, certain trade-off points could not be achieved. In the extreme case of $T = 1$, the controller either did not follow the workload, resulting into a very high SLA cost and close to zero resource and reconfiguration costs or the solver failed to find a solution. For $T = 2$ (see Figure 6.9b) the controller fails to achieve a reconfiguration cost within (100,700) range or an SLA cost within (100,1000) range. To verify this, we tuned the cost coefficients several times (not represented in figure 6.9b) but we were unable to achieve any cost within these ranges. With $T = 7$, the controller is roughly able to achieve all the regions that the optimal controller is able to achieve (see Figure 6.9g and 6.9i).

6.6.3 Scalability and Computational Complexity

The computational complexity of our approach can be quantified in terms of the number of variables passed to the optimization solver. The number of variables passed to the solver has a linear relationship with the number of hosts, the number of services, and the size of lookahead horizon. For a small data center with 10 hosts, 10 services, and the lookahead horizon of 10 seconds we had roughly 1400 variables passed to the optimizations engine. Solving an instance of this problem on a quad core i5 Intel processor using Matlab and SeDuMi solver roughly took 4 to 5 seconds. We expect that, for a large service center with 10000 hosts and lookahead horizon of 5 (i.e. five minutes) and a dedicated server for solving the optimization problem it should not take more than 5 minutes to solve an instance of the problem. Since the time step granularity at that scale is at least 10 minutes, the algorithm should be able to finish its calculation in before moving on to the next step.

6.7 Summary

In this chapter, we introduced a new optimization model and a simple fast algorithm for the optimal service placement (OSP) of a set of N-tier software systems, subject to changes in the workloads, SLAs, and the administrators preferences. The model captures the reconfiguration cost, and the algorithm uses the MPC paradigm.

We first formulated the OSP as a stochastic model predictive control problem. We enumerated several subtleties with the formulated problem. These subtleties include nonlinear queuing model and nonlinear and discontinuous cost elements. We then addressed these subtleties and proposed a fast solution to the proposed problem. The solution deals with the non-linearity of the LQM, non-linearity of the resource cost and reconfiguration cost, and the stochastic nature of the workloads. A solver friendly format of the solution was also derived. We also provided an algorithm that takes contention into account more accurately. However, it requires solving the actual LQM multiple times to derive an OSP solution for a single step.

The described experiments validated our hypothesis that model predictive approaches perform better than simple stepwise optimization when the objective functions are long term and when reconfiguration is taken into account.

7 Summary and Conclusion

In this thesis, we presented three contributions to the dynamic model-based resource management of multi-service information systems. In this chapter we summarize our contributions, describe the limitations of our work, and discuss the assumptions we made. We then discuss issues for future work.

7.1 Contributions

In the first contribution (Chapter 4) we targeted improving the estimation of performance parameters in multi-class Layered Queuing Models. We proposed a tracking approach which requires less monitoring overhead and computational complexity. Instead of estimating for individual classes it identifies the performance parameters of groups of classes which are relatively similar. We proposed an algorithm that finds the appropriate number of clusters given a desired estimation accuracy. In the experiments, we used a web application and modelled the application's URLs as separate classes. We showed the usefulness of the K-means algorithm. We also

showed that our Extended Kalman filter could track hidden states successfully, and that the accuracy of the filter rises as it tries more classes and re-estimates the service demands.

In chapter 5, we demonstrated the advantage of dynamically-tuned empirical models relative to static ones, in model based optimization of a private cloud where applications are clustered across a known, homogeneous set of physical machines. In this optimization, we modified the resource shares of the applications, in order to minimize their SLA violations. The main contribution of this work was dynamically tracking parameters of the empirical models (for each application) within a global optimization loop. These models update themselves at runtime in order to adapt to aspects in the environment not captured in the initial model specifications. This results in more accurate models being passed to the optimizer, allowing for better resource utilization on a global scale. Another contribution in this chapter was a formulation of the optimization problem based on the adaptive model. The optimization maximizes a utility function defined over SLAs and shows the benefits of the use of adaptive models at runtime.

In chapter 6, we introduced a new optimization model and fast algorithm for optimal service placement (OSP) of a set of N-tier software systems, subject to changes in the workload, SLA, and administrators preferences. The OSP problem was posed as a stochastic model predictive control problem. We enumerated several

subtleties with the formulated problem such as non-linearity of the LQM, resource cost, and reconfiguration cost and discussed how to overcome them.

We proposed a fast solution to the proposed problem that deals with these issues. This solution performs well if the system is not saturated. We could easily force this condition (i.e. the system being lightly loaded) so that the solution pushes the system to an un-saturated region. A more precise algorithm with more computational overhead is also proposed. However, the precise algorithm requires solving the actual LQM multiple times to derive an OSP solution for a single step.

The experiments validated our hypothesis that the model predictive approach performs better than the simple stepwise optimization when the objective functions are long term and when reconfiguration is taken into account.

7.2 Limitations and Future Work

In chapter 4 (the first contribution), our goal was to minimize both the introduced modeling error and the complexity of the model during estimation. We used the maximum acceptable error value of 8% in the experiments, to control the balance between the introduced error and complexity of the model. However, this control was done implicitly. In future work, one can model the relationship between the computation complexity and the modelling error explicitly, and try to find the optimal number of clusters which minimizes both.

Based on our investigation in chapter 4, the Bayesian approach has a number of benefits over regression-based techniques. First, Bayesian estimation provides more flexibility in terms of choosing different models. It also provides more control over different aspects of the estimation. However, Bayesian estimation is very demanding with respect to the number of needed measurements.

Another research question regarding the first contribution is comparing the moving horizon regression-based estimation techniques for service demand estimation with the Bayesian approach in terms of scalability. We suspect that because of their simplicity, they would actually outperform the Bayesian estimation in terms of scalability.

In chapter 5 (the second contribution), the relationship between the number of simulated VMs and the optimization time per step was considered, and a non-linear but polynomial relation was observed. While this could pose a problem for our proposed approach (i.e., when working with large numbers of VMs) a possible solution would be to split the problem into subproblems and solve them individually. Future work will involve implementing the optimization algorithm in a distributed manner in which applications interact in a peer-to-peer fashion, to determine how much resource should be allocated to each application. One can also investigate solving the dual optimization problem, which is more suitable for mapping to an agent-oriented optimization approach (see [45, 46]).

One possible future work for chapter 6 would involve investigating the effect of modeling on the performance of the control. For example, one can consider different models of non-stationary autonomous processes for the workload, and see if these models result into different performances in the overall control. An interesting aspect would be to investigate if under certain workloads (e.g. random walk) the model predictive control is not really needed (i.e. the one-step-ahead optimization would be enough). One can also model the workload as a dynamic system whose inputs are driven by real-world entities such as time (i.e. time of the day, day of the week, or month of the year).

Another future investigation for the third contribution would be testing different formats of the objective function. For example, one could test the behavior of the controller under quadratic reconfiguration cost. According to our experience, changing the format of the cost function would introduce major changes in the behavior of the controller. For example, switching from first norm to the second norm for the reconfiguration cost makes the controller spread the reconfigurations over time in a smooth fashion. On the other hand, our experience shows that solving control problems using convex optimization solvers would result in a non-sparse deployment which is not desired. One can test if the result obtained from other types of solvers (for example a customized one that is written from scratch) would have different properties in terms of the density of the deployment.

7.3 Summary

In this chapter, we provided the summary of this dissertation. We described the tree main contributions of the thesis. We also explained the limitations of our approaches and provided possible directions for the future work.

8 Appendix A: LQN Solution

This chapter describes one of the solutions to Layered Queuing Networks (LQN). In LQN, two queuing networks, associated with hardware and software, coexist at the same time and are solved in parallel. Since both the hardware and the software layers affect one another, a fixed point solution for LQN can only be found for the whole network [79] by iterating among the layers of the QNs. The parameters exchanged between the layers are:

1. From the software to the hardware: the number of blocked users (or processes) at the software level (waiting for a software resource) $\vec{B} = (B_1, \ldots, B_C)$, where B_c denotes the number of class c processes blocked for a software resource.

2. From hardware to software: the response time at each hardware queuing centre.

The iterative algorithm, taken from Menascé's work [68, 69], is as follows:

- Step 1 - Initialization:

$$D_{c,s} \leftarrow \sum_{k} D_{c,s,k} \tag{8.1}$$

$$D_{c,k} \leftarrow \sum_{s} D_{c,s,k} \tag{8.2}$$

$$\vec{B^0} \leftarrow \vec{0} \text{ initial value for B} \tag{8.3}$$

$$i \leftarrow \text{iteration counter} \tag{8.4}$$

- Step 2 - Solve the SQN with $D_{c,s}$ as the service demands and $\vec{N^s}$ as customer population.

- Step 3 - Compute the average number of blocked processes per class B^i; that is the average number of processes waiting in the software queue. This number is taken from $\vec{B^i} = (\sum_s B_{1,s}, \ldots, \sum_s B_{C,s})$, where $B_{c,s}$ is the average number of class c processes in the waiting line for software resource s in the SQN.

- Step 4 - Solve the HQN with $D_{c,k}$ as service demands and $\vec{N^h} = \vec{N^s} - \vec{B^i}$ as the population vector.

- Step 5 - Adjust the service demands at the SQN to account for contention at the physical resources:

$$D_{c,s} \leftarrow \sum_{k} \frac{D_{c,s,k}}{D_{c,k}} R_{c,k}(\vec{N^h}) \tag{8.5}$$

- Step 6 (convergence check step): If $\max|(B_c^i - B_c^{i-1})/B_c^i| > \xi$ then $i \leftarrow i + 1$ and goto step 2.

Note that in the Menascé algorithm (above) the throughput of the system depends saturation levels of software services (e.g. thread pools) and hardware resources. The requests have to wait for an empty thread before they can proceed to use hardware layer resources. A thread pool saturated due to a low resource multiplicity (number of active threads) limits the number of users in the hardware layer, and decreases the overall throughput. If the software resources are not saturated and impose no queuing, the throughput of the system will be almost equal to throughput of underlying hardware. In addition, the saturation level of software resources depends on departure rate in hardware level. A saturated hardware resource also decreases the overall throughput, and saturates both hardware and software queues. The bottleneck may switch from hardware resources to software resources and vice versa with a change in the configuration and the multiplicity of resources [1, 21].

Bibliography

[1] Application Performance Evaluation and Resource Allocator. http://www.ceraslabs.com/projects/performance-engineering [online March 2012].

[2] Autonomic computing in Canadian academia. http://www.ibm.com/developerworks/library/ac-canada/index.html [online August 2011].

[3] collectd The system statistics collection daemon. http://collectd.org/ [online August 2011].

[4] Discrete linear quadratic regulator. http://www.mathworks.com/help/control/ref/dlqr.html[online August 2011].

[5] IBM Tivoli Monitoring. http://www.ibm.com/software/tivoli/products/monitor [online June 2014].

[6] Java Management Extensions (JMX). http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html [online August 2011].

[7] Nagios IT Infrastructure Monitoring. http://www.nagios.org/ [online August 2011].

[8] radiantGrid platform. http://www.radiantgrid.com/feature-grid/ [online August 2011].

[9] TPC-W java implementation. http://mitglied.multimania.de/jankiefer/tpcw/[online August 2011].

[10] TPW-C benchmark specification.

[11] S. Abdelwahed, N. Kandasamy, and S. Neema. A control-based framework for self-managing distributed computing systems. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 3–7. ACM, 2004.

[12] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, pages 80–96, 2002.

[13] S. Aiber, D. Gilat, A. Landau, N. Razinkov, A. Sela, and S. Wasserkrug. Autonomic self-optimization according to business objectives. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 206–213. IEEE, 2004.

[14] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *Network, IEEE*, 14(3):30 –37, may. 2000.

[15] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009. `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html`.

[16] O. Babaoglu, M. Jelasity, and A. Montresor. Grassroots approach to self-management in large-scale distributed systems. *Unconventional Programming Paradigms*, pages 286–296, 2005.

[17] S. Babu and J. Widom. Continuous queries over data streams. *ACM Sigmod Record*, 30(3):109–120, 2001.

[18] E. Badidi, L. Esmahi, and M.A. Serhani. A queuing model for service selection of multi-classes QoS-aware web services. In *Third IEEE European Conference on Web Services (ECOWS)*, page 9, nov 2005.

[19] Jia Bai and Sherif Abdelwahed. Efficient algorithms for performance management of computing systems. In *Fourth International Workshop on Feed-*

back Control Implementation and Design in Computing Systems and Networks FeBID. IEEE, 2009.

[20] R.K. Balan, M. Satyanarayanan, S.Y. Park, and T. Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 273–286. ACM, 2003.

[21] Cornel Barna, Marin Litoiu, and Hamoun Ghanbari. Model-based performance testing: NIER track. In *Proceeding of the 33rd international conference on Software engineering*, pages 872–875, Waikiki, Honolulu, HI, USA, 2011. ACM.

[22] Mohamed N. Bennani and Daniel A. Menascé. Assessing the robustness of self-managing computer systems under highly variable workloads. *Autonomic Computing, International Conference on*, 0:62–69, 2004.

[23] V. Bhat, M. Parashar, et al. Enabling self-managing applications using model-based online control strategies. In *2006 IEEE International Conference on Autonomic Computing*, pages 15–24. IEEE, 2006.

[24] E. Brookner. *Tracking and Kalman filtering made easy*. John Wiley & Sons, 1998.

[25] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[26] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 2010.

[27] J.W. Cangussu, K. Cooper, and C. Li. A control theory based framework for dynamic adaptable systems. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1546–1553. ACM, 2004.

[28] F. Chang and V. Karamcheti. Automatic configuration and run-time adaptation of distributed applications. In *hpdc*, page 11. Published by the IEEE Computer Society, 2000.

[29] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J.S. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation-Volume 6*, pages 16–16. USENIX Association, 2004.

[30] M. Courtois and M. Woodside. Using regression splines for software performance analysis. In *Proceedings of the 2nd international workshop on Software and performance*, pages 105–114. ACM, 2000.

[31] Ronald P. Doyle, Jeffrey S. Chase, Omer M. Asad, Wei Jin, and Amin M. Vahdat. Model-based resource provisioning in a web service utility. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, pages 5–5, Seattle, WA, 2003. USENIX Association.

[32] William Dumouchel and Fanny O'Brien. Integrating a robust option into a multiple regression computing environment. *Institute for Mathematics and Its Applications*, 36:41, 1991.

[33] B. Ensink and V. Adve. Coordinating Adaptations in Distributed Systems. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 446–455. IEEE Computer Society, 2004.

[34] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115, 1997.

[35] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, pages 37–46, 2002.

[36] Daniel F. García and Javier García. TPC-W e-commerce benchmark evaluation. *Computer*, 36(2):42–48, 2003.

[37] D. Garlan, S.W. Cheng, A.C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[38] Hamoun Ghanbari, Marin Litoiu, Murray Woodside, Tao Zheng, Johnny Wong, and Gabriel Iszlai. Tuning tracking of performance model parameters using dynamic job classes. In *Proceedings of the second ACM/SPEC International Conference on Performance Engineering (ICPE 2011), to appear.* ACM, 2011.

[39] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Capacity management and demand prediction for next generation data centers. In *Proceedings of the International IEEE Conference on Web Services*, pages 43–50, Salt Lake City, Utah, USA, July 2007. IEEE Computer Society.

[40] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Proceedings of the IEEE 10th International Symposium on Workload Characterization (IISWC'07)*, volume 0, pages 171–180, Boston, MA, 2007. IEEE Communications Society.

[41] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. `http://cvxr.com`, March 2012.

[42] Brian Hayes. Cloud computing. *Communications of the ACM*, 51(7):9–11, 2008.

[43] A. Huang and P. Steenkiste. Building self-configuring services using service-specific knowledge. In *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on*, pages 45–54. IEEE, 2004.

[44] A.C. Huang and P. Steenkiste. Network-sensitive service discovery. *Journal of Grid Computing*, 1(3):309–326, 2003.

[45] Peijie Huang, Hong Peng, Piyuan Lin, and Xuezhen Li. Macroeconomics based grid resource allocation. *Future Generation Computer Systems*, 24(7):694 – 700, 2008.

[46] Hesam Izakian, Ajith Abraham, and Behrouz Tork Ladani. An auction method for resource allocation in computational grids. *Future Generation Computer Systems*, 26(2):228 – 235, 2010.

[47] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman

filters. In *Proceedings of the 6th international conference on Autonomic computing (ICAC '09)*, pages 117–126, Barcelona, Spain, June 2009. ACM.

[48] N. Kandasamy, S. Abdelwahed, and JP Hayes. Self-optimization in computer systems via on-line control: application to power management. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 54–61. IEEE, 2004.

[49] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York, NY, USA, 1990.

[50] J Kephart, JO Kephart, DM Chess, Craig Boutilier, Rajarshi Das, Jeffrey O Kephart, and William E Walsh. An architectural blueprint for autonomic computing. *IEEE internet computing*, 18(21), 2007.

[51] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):4150, 2003.

[52] Stephan Kraft, Sergio Pacheco-Sanchez, Giuliano Casale, and Stephen Dawson. Estimating service resource consumption from response time measurements. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '09, pages 48:1–48:10, Brussels, Belgium, 2009. ICST.

[53] V. Kumar, BF Cooper, Z. Cai, G. Eisenhauer, and K. Schwan. Resource-Aware Distributed Stream Management Using Dynamic Overlays. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 783–792. IEEE, 2005.

[54] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1984.

[55] ED Lazowska, J Zahorjan, GS Graham, and KC Sevcik. Quantitative system performance, computer system analysis using queuing network models, by prentice-hall. *Englewood Cliffs, NJ*, page 117, 1984.

[56] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen. A scalable solution to the multi-resource QoS problem. In *Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE*, pages 315–326. IEEE, 1999.

[57] H. Li, G. Casale, and T. Ellahi. SLA-driven planning and optimization of enterprise applications. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 117–128. ACM, 2010.

[58] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. Performance model driven QoS guarantees and optimization in clouds. In *Proceedings of the 2009*

ICSE Workshop on Software Engineering Challenges of Cloud Computing, pages 15–22. IEEE Computer Society, 2009.

[59] J. Z. Li, J. Chinneck, M. Woodside, and M. Litoiu. Fast scalable optimization to configure service systems having cost and quality of service constraints. In *Proceedings of the 6th international conference on Autonomic computing*, pages 159–168. ACM, 2009.

[60] Jim Zhanwen Li. *Fast Optimization for Scalable Application Deployments in Large Service Centers*. PhD thesis, Carleton University, 2011.

[61] Aristidis Likas, Nikos Vlassis, and Jakob J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition Society*, 36(2):451–461, February 2003.

[62] M. Litoiu, M. Woodside, and T. Zheng. Hierarchical model-based autonomic control of software systems. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, May 2005.

[63] John DC Little. A proof for the queuing formula: L= λw. *Operations research*, 9(3):383–387, 1961.

[64] T. K. Liu, S. Kumaran, and Z. Luo. Layered queuing models for enterprise

Javabean applications. In *Proc. 5th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 4–7, 2001.

[65] Z. Liu, L. Wynter, C. H. Xia, and F. Zhang. Parameter inference of queueing models for it systems using end-to-end measurements. *Performance Evaluation, Elsevier*, 63(1):36–60, 2006.

[66] J.P. Loyall, R.E. Schantz, J.A. Zinky, and D.E. Bakken. Specifying and measuring quality of service in distributed object systems. In *Object-Oriented Real-time Distributed Computing, 1998.(ISORC 98) Proceedings. 1998 First International Symposium on*, pages 43–52. IEEE, 1998.

[67] D Menascé. Computing missing service demand parameters for performance models. In *Proceedings of the Thirty-fourth International Computer Measurement Group Conference, December*, pages 7–12, 2008.

[68] D.A. Menascé. Simple analytic modeling of software contention. *ACM SIGMETRICS Performance Evaluation Review*, 29(4):24–30, 2002.

[69] D.A. Menasce, V.A.F. Almeida, L.W. Dowdy, and L. Dowdy. *Performance by design: computer capacity planning by example*. Prentice Hall, 2004.

[70] Daniel A Menasce. Composing web services: A QoS view. *Internet Computing, IEEE*, 8(6):88–90, 2004.

[71] Daniel A Menascé, Virgílio AF Almeida, and Larry W Dowdy. *Capacity planning and performance modeling: from mainframes to client-server systems*. Prentice-Hall, Inc., 1994.

[72] Daniel A Menascé, Virgilio AF Almeida, Rodrigo Fonseca, and Marco A Mendes. A methodology for workload characterization of e-commerce sites. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 119–128. ACM, 1999.

[73] M. Mesnier, E. Thereska, G.R. Ganger, D. Ellard, and M. Seltzer. File classification in self-* storage systems. 2004.

[74] Klaus Müller and Tony Vignaux. SimPy: Simulating Systems in Python. *ONLamp.com Python DevCenter*, February 2003.

[75] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker. Agile application-aware adaptation for mobility. *ACM SIGOPS Operating Systems Review*, 31(5):276–287, 1997.

[76] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. CPU demand for web serving: Measurement analysis and dynamic estimation. *Performance Evaluation*, 65(6-7):531–553, 2008.

[77] T. Patikirikorala, L. Wang, A. Colman, and J. Han. Hammerstein–Wiener

nonlinear model based predictive control for relative QoS performance and resource management of software systems. *Control Engineering Practice*, 2011.

[78] D. C. Petriu. Approximate mean value analysis of client-server systems with multi-class requests. In *Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 77–86. ACM New York, NY, USA, 1994.

[79] D. C. Petriu and C. M. Woodside. Approximate MVA from markov model of software client/server systems. In *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing.*, pages 322–329, Dallas, Texas, 1991. IEEE Computer Society.

[80] D. C. Petriu and C. M. Woodside. Approximate mean value analysis based on markov chain aggregation by composition. *Linear Algebra and its Applications*, 386:335–358, 2004.

[81] S. Ramesh and H. G. Perros. A multi-layer client-server queueing network model with synchronous and asynchronous messages. In *Proceedings of the 1st international workshop on Software and performance*, pages 107–119. ACM New York, NY, USA, 1998.

[82] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, 2011.

[83] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):535–545, july 2009.

[84] J. Rolia and V. Vetland. Parameter estimation for performance models of distributed application systems. In *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*, page 54. IBM Press, 1995.

[85] J. Rolia and V. Vetland. Correlating resource demand information with ARM data for application services. In *Proceedings of the 1st international workshop on Software and performance*, pages 219–230. ACM New York, NY, USA, 1998.

[86] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Transactions on Software Engineering*, 21(8):689–700, 1995.

[87] Abraham Seidmann, Paul J Schweitzer, and Sarit Shalev-Oren. Computerized closed queueing network models of flexible manufacturing systems: A comparative evaluation. *Large Scale Systems*, 12:91–107, 1987.

[88] Abhishek B Sharma, Ranjita Bhagwan, Monojit Choudhury, Leana Golubchik, Ramesh Govindan, and Geoffrey M Voelker. Automatic request categorization in internet services. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):16–25, 2008.

[89] A.A. Soror, U.F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. *ACM Transactions On Database Systems*, 35(1):Article 7, 2010.

[90] Jos F Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625–653, 1999.

[91] H. Tanizaki. *Nonlinear filters: estimation and applications*. Springer, 1996.

[92] Gerald Tesauro and Jeffrey O. Kephart. Utility functions in autonomic systems. In *Proceedings of the First International Conference on Autonomic Computing*, pages 70–77. IEEE Computer Society, 2004.

[93] W Tsai, Chun Fan, Yinong Chen, and R Paul. DDSOS: a dynamic distributed service-oriented simulation framework. *Simulation Symposium, 2006. 39th Annual*, 2006.

[94] W.T. Tsai, Zhibin Cao, Xiao Wei, Ray Paul, Qian Huang, and Xin Sun.

Modeling and simulation in service-oriented software development. *Simulation*, 83(1):7–32, 2007.

[95] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *The IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC*, pages 153–158, Alberta, Canada, 2000.

[96] Y.M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H.J. Wang, C. Yuan, and Z. Zhang. STRIDER: A black-box, state-based approach to change and configuration management and support. In *Proceedings of the 17th USENIX conference on System administration*, pages 159–172. USENIX Association, 2003.

[97] P. K. Watson. Kalman filtering as an alternative to ordinary least squares some theoretical considerations and empirical results. *Empirical Economics*, 8:71–85, 1983. 10.1007/BF01973191.

[98] G. Welch and G. Bishop. An introduction to the Kalman filter. *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 1995.

[99] C. J. Wild. *Nonlinear regression.* Wiley, New York, 1989.

[100] M. Wolf, Z. Cai, W. Huang, and K. Schwan. SmartPointers: Personalized

Scientific Data Portals In Your Hand. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 20–20. IEEE, 2002.

[101] M. Woodside, T. Zheng, and M. Litoiu. The use of optimal filters to track parameters of performance models. In *Quantitative Evaluation of Systems, 2005. Second International Conference on the*, pages 74–83, 2005.

[102] J. Xu, A. Oufimtsev, M. Woodside, and L. Murphy. Performance modeling and prediction of enterprise JavaBeans with layered queuing network templates. In *Proceedings of the 2005 conference on Specification and verification of component-based systems (SAVCBS '05)*, Lisbon, Portugal, 2005. ACM.

[103] Jing Xu, Alexandre Oufimtsev, Murray Woodside, and Liam Murphy. Performance modeling and prediction of enterprise JavaBeans with layered queuing network templates. *ACM SIGSOFT Software Engineering Notes*, 31(2):5, 2006.

[104] L. Zhang, C. Xia, M. Squillante, and W. N. Mills III. Workload service requirements analysis: A queueing network optimization approach. In *Proceedings of 10th IEEE International Symposium on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems*, Washington, DC, 2002. IEEE Computer Society.

[105] Q. Zhang, L. Cherkasova, and E. Smirni. A Regression-Based analytic model

for dynamic resource provisioning of Multi-Tier applications. In *Proceedings of the 4th IEEE International Conference on Autonomic Computing*, page 27, Jacksonville, Florida, 2007. IEEE Computer Society.

[106] Qi Zhang, Mohamed Faten Zhani, Shuo Zhang, Quanyan Zhu, Raouf Boutaba, and Joseph L Hellerstein. Dynamic energy-aware capacity provisioning for cloud computing environments. In *Proceedings of the 9th international conference on Autonomic computing*, pages 145–154. ACM, 2012.

[107] Qi Zhang, Quanyan Zhu, Mohamed Faten Zhani, and Raouf Boutaba. Dynamic service placement in geographically distributed clouds. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 526–535. IEEE, 2012.

[108] T. Zheng, M. Woodside, and M. Litoiu. Performance model estimation and tracking using optimal filters. *IEEE Transactions on Software Engineering*, 34:391–406, 2008.

[109] T. Zheng, J. Yang, M. Woodside, M. Litoiu, and G. Iszlai. Tracking time-varying parameters in software systems with extended kalman filters. In *Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, page 345. IBM Press, 2005.

[110] Tao Zheng, Marin Litoiu, and Murray Woodside. Integrated estimation and

tracking of performance model parameters with autoregressive trends. In *Proceedings of the 2Nd ACM/SPEC International Conference on Performance Engineering*, ICPE '11, pages 157–166, New York, NY, USA, 2011. ACM.