Model-Based Dynamic Resource Management For Multi Service Information Systems

Hamoun Ghanbari

Abstract

Cloud computing provides the computational power of data centers to a large user community over the web. In Software as a Service (SaaS) and Platform as a Service (PaaS) forms of the cloud, multiple high-level services and end-user applications are hosted on the cloud provider's available resources. A major problem for SaaS and PaaS cloud providers, is associating the optimal amount of physical hardware to services. The optimality of this association is measured in terms of meeting the promised Quality of Service (QoS) for the applications while minimizing the overall infrastructure cost. We target solving two variations of this problem using model-based control. We model the cloud infrastructure, the services, and the applications using the Layered Queuing Models (LQM). To build and maintain such models we use an existing estimation technique called Kalman Filtering (KF). In our first contribution, we improve the convergence of the existing Kalman Filter based estimation for large LQM models. This is done by dynamically clustering the applications into a smaller set. This improves the convergence of the estimation while keeping the modeling error at an acceptable level. The second problem involves the allocation of limited resources to a set of applications by tuning the fraction of resource capacity allocated to each application.

The optimization used in this case, works best when there is no restriction, guideline, or associated cost on the number of reconfigurations. In presence of such restrictions, this approach can create overhead and perturbations for the running applications. To alleviate the above problem, in our third contribution, we propose a dynamic service placement algorithm that considers the trashing cost in calculating the optimal configuration using Model Predictive Control (MPC). Through experiments, we validated our hypothesis that model predictive approach performs better than the simple stepwise optimization when the objective functions are long term and when trashing is taken into account.

1 Introduction

Cloud computing provides the computational power of data centers (e.g., network, storage, computational devices, and services) to a large user community over the web. A cloud is formed by an interconnected set of datacenters. A data center environment is composed of a communication system, servers, and astorage subsystem. These computing resources are hosted in controlled environments and under centralized management. Currently there are three types of cloud computing offerings: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS multiplexes the hardware layer and offers computing services such as storage, CPU and memory to users. In SaaS and PaaS, multiple higher-level services and end-user applications are hosted on Cloud provider's available resources. A PaaS provider is an enterprise that is responsible for leasing application environments or programmable services to customers for various durations of time. Customers of PaaS could deploy their own programs on purchased platforms from a PaaS provider. SaaS type of Cloud provides instances of software and applications that are typically installed in businesses' computer networks or personal computers. Examples of this software include customer relationship management (CRM), accounting, invoicing, human resource management (HRM), and content management (CM). Because SaaS and PaaS host services, they are sometimes referred to as Multi Service Information System (MSIS)[9], application service centers, or service centers in short.

A major problem for SaaS and PaaS cloud providers is associating the optimal amount of physical hardware (including CPUs, networks, etc.) to services. The amount of the physical hardware 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)[20]. For OSP, the optimality is measured in terms of meeting the promised Quality of Service (QoS) mentioned in Service Level Agreement (SLA)[1] while minimizing the overall cost. SLAs include a set of performance metrics experienced by applications or classes of customers[2]. Performance metrics usually represent time behavior of the services such as average throughput, mean response time, total percentage of requests rejected or not handled within certain time limit. In a PaaS and SaaS, the cost is associated with

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

[2]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.

the number of active hardware components contributing to the cost of electricity and cooling.

In the remainder of this section, we present the fundamentals and assumptions of our work. Subsection 1.1 presents the model-based approach to resource management. Subsection 1.2 presents our model for the service center.

1.1 Model Based Resource Management Architecture

A solution to an OSP problem is a sequence of feasible allocation actions over time that maximizes a defined performance criterion (or an objective function) regarding SLAs[3]. In our approach, these allocation decisions are made at each time-step based on the information available from the system up to that time. A controller constantly considers the most recent monitored data for calculating the proper actions. Thus, the original plan is adjusted according to the new observation samples of the environment in the past step. This is called the feedback based scheme or closed loop control.

The model-based feedback loop we propose is based on the IBM's MAPE-K loop architecture, which is composed of the following four subsystems: a monitor, an analyzer, a planner and an execution subsystem. Figure 1 represents the schematic view of the IBM's MAPE-K loop.

The Monitoring subsystem is responsible for measuring inputs, and outputs of the managed system, quantifying them, sometimes aggregating them, and keeping them as a history. In our case, the monitored metrics are the service specific performance metrics for different types of server processes

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

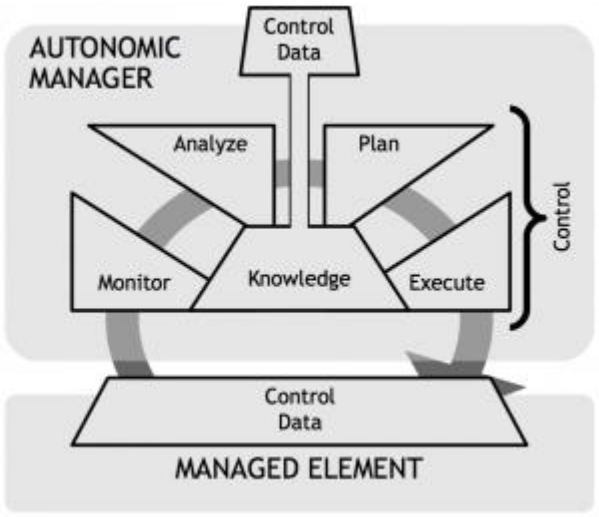

Figure 1: Schematic view of the IBM's MAPE-K loop.

(e.g. load balancer, web server, application server, and database server).

The analyzer subsystem, identifies and tunes a model of the system under management, and estimates the unobservable portion of system state. The model enables the planner to project the system's behavior and the state under different actions in the future. Queuing theory based models [12, 13, 2] and Layered Queuing Models (LQM) [15, 14] developed upon Mean Value Analysis (MVA) of queuing networks have been widely used to capture the behavior of multi-tier distributed applications [10, 19, 5, 11]. They can be utilized to describe the expected performance of service centers (i.e. response time and throughput) in relation to various inputs. A major concern is the ability to synchronize the model based on the observed behavior of the system. In our case, the analyzer uses statistical techniques to maintain an up-to-date LQM.

A planner uses the model to rapidly explore multiple decisions and find near-optimal solution towards a goal [10, 1].

A execution subsystem applies the derived control actions to the system.

1.2 The LQM Based Cloud Model

A service center can be modeled by a set of queues where devices are mapped to queuing or delay centers. Based on the physical structure of the data center and the service invocations, one can build a queuing networks model. Then, a general Mean Value Analysis (MVA) of the model can be used to calculate the performance characteristics of the system.

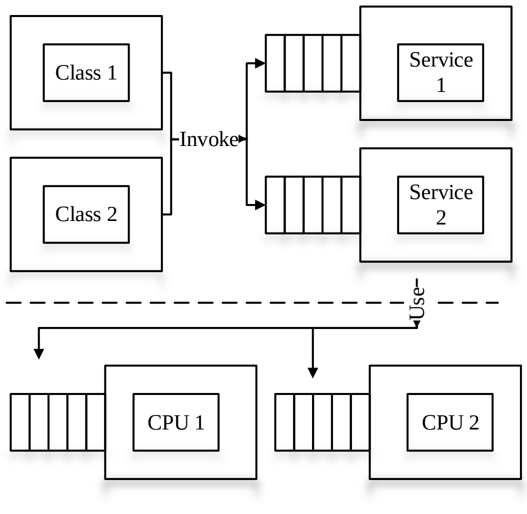

Figure 2: A simplified view of LQM, which is used to model a service center. The dashed line denotes the separation between software and hardware layers.

Workload of a service is driven by its external users hence usually quantified by number of users and think time or request arrival rate, which is measured in requests per seconds. Each group of users uses 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 queuing 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 with a fixed population. CPU is the main resource accessed by services. Other resources such as hard disks and network are used from CPU.

A simplified view of LQM which is used to model a service center is presented in figure 2. Figure 3 demonstrates an example of a service center, composed of 14 services (denoted by s1 to s14), accessed by 2 classes of users (the two rectangles at the top). The figure represents the call graph of the services, the think times (Z_c), the number of users (N_c) of each class over time, and the invocation numbers within services (i.e. the numbers placed beside the arrows). As an example, here we give one interpretation for the model. Assume s3 is an accounting application, and s6 is a database service. Every request of the accounting application on average invokes the database service 2.5 times. There are two organizations using the accounting application. Initially organization1 has 250 users and organization2 has 100 users. On each round of visit to the service center[4] organization1 invokes application once and organization2 invokes the application twice.

[4]The model is flexible in terms of defining the 'round of visit'.

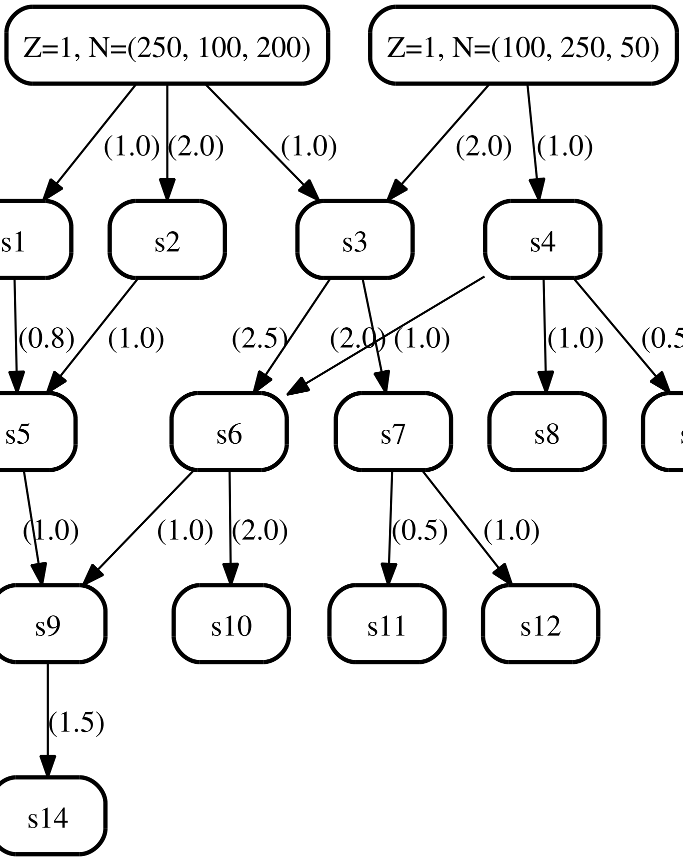

Figure 3: An example small scale service center: the call graph of services and classes, the think times Z and number of users N of each class over time (e.g. 250, 100, 200 is the number of users for the next three steps for the class 1). The edges between services represent invocations between services, and the numbers placed beside these ages represent the number of invocations. The edges between classes and services represent visit ratios.

1.2.1 Application scaling and optimization through service replication

An important tool in controlling the performance of applications is to adjust the amount of resource given to their services. Through replication, each service can be distributed across multiple physical machines. Each instance of a service is then called a replica. Each replica will handle a portion of the original service's workload. The actual demand on each resource is obtained by taking into account this association.

One can adjust the load on each replica by distributing the accesses to replicas in a desired proportion. In fact, in our third contribution, we use optimization to determine the number of replicas, allocation of the replicas to physical machines, and the amount of load on each replica.

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

The LQM model described earlier, needs to be built based on the measurement data. This requires that the service demands and the workload parameters be estimated for each class of users. Bayes filters have been successfully applied to perform this task [17, 18, 21]. Despite effectiveness, an important issue in using the Bayesian approach is convergence. Bayes filters do not converge to a solution unless we have more measured parameters than estimated state parameters. Therefore, the number of classes should be reduced to a certain size, hugely reducing the scalability of the estimation to model large data centers.

We propose the clustering of user classes into a smaller set with lower cardinality, therefore increasing the scalability of Bayesian performance model estimation. By decreasing the number of classes, we also reduce the amount of computation the estimator has to perform. However, by grouping the classes we introduce some modeling error[5]. That is the reason we cannot cluster all the classes as into one group. We propose an algorithm to determine the best choice of the number of clusters and the grouping of classes into these clusters, in a way that the estimation converges at the same time we have a low modeling error. The major contribution of our approach is, thus, dynamic clustering of user classes, to improve the convergence of the filter and yet leave enough classes to satisfy the accuracy criterion.

In the paper [5], we validated the method with a set of experiments on a deployed web application. The first set of experiments were tracking the service demand of a web application. We tested if the algorithm could maintain the same upper bound on the modeling error by dynamically changing the number of clusters over the experiment. The test was passed successfully; for a given modeling error percentage the algorithm could maintain the same modeling error by adjusting the number of clusters. For example, it could reach 17% error using only 60% of the original classes.

[5]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 parametrized statistical distributions. 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 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.

We also performed the same test for a case where the service demands change at different rates and phases in short run. The system under test was a simulated web based e-commerce site. We observed that our estimation and classification algorithm works quite well, since it is able to keep the error below 8% with the smallest number of clusters and acceptable frequency of re-clustering.

To the best of our knowledge, the combination of dynamic clustering and parameter estimation has never been investigated before. The only related work in terms of final goal is [16]. They try to categorize requests based on their resource usage characteristics. However, they ignore the prior knowledge about requests in the system and classify the requests on the fly. Thus, their utilized approach is also different, as they use a machine learning technique called Independent Component Analysis (ICA).

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 where applications are deployed on virtual machines (VMs). In a virtualized data center, this 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. Necessary for this optimization, is a model of the application service center. The model can be either first principle (e.g. based on queuing theory) or be driven empirically from a data set.

A portion of the related work on optimal deployment such as [8, 7] assumes that the system follows an accurate first principle model. They suggest using a filter-based approach such as [22] to adaptively estimate the parameters of this model. However, there is no guarantee that these first principle models accurately represent a service center, since it is usually very hard to model all aspects of a service center such as software contention, concurrency, remote calls, limited amount of memory, and limited amount of network bandwidth.

On the other end, there is a major problem with the empirical models. They do not propose a way to tune the models based on the measurement obtained during runtime. When applications are deployed in the production environment, the original models start to show inaccuracy. This is mainly because the deployment conditions are different from the test environments, which are used to build the training data sets.

Our contribution is to increase the accuracy of the empirical models by tuning them dynamically using the measurement data at runtime on the fly. We also trace this 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 the one that is not tuned. The other differences between our approach and the related work are the use of the decomposed models, and a custom optimization routine to optimize the defined utility functions.

In [6], we successfully tested the contribution. We simulated a small cloud configuration with multiple instances of applications deployed on physical machines using CloudSim [3]. We showed static non-tuned models of the applications results in a suboptimal allocation decision for some applications and leads to failure in meeting their SLAs. In contrast, using dynamically tuned models results in a more efficient resource allocations being made and better commitment to SLAs and better utility. We have to mention that, in the experiments, to lower the complexity of the experiments, we assumed that the capacity of the hosts, the workload intensities, and the service demands are either monitored or directly obtained; thus, the only quantities to be estimated (i.e. tuned) are the regression model coefficients.

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

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

To alleviate the above problem, we propose a dynamic service placement algorithm that considers the trashing cost in calculating the optimal configuration using Model Predictive Control (MPC). It 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 validated our hypothesis that the model predictive approach performs better than the simple stepwise optimization (e.g. an optimization that does not consider the future time such as the optimization performed in the second contribution) when the objective functions are long term and when trashing is taken into account. The result is represented in the paper [4] which is in the review process.

The most notable related work to this contribution is [20]. It targets optimal service placement in geographically distributed clouds with reconfiguration cost taken into account through MPC. However, it has two main differences from our proposal: (i) it uses a single class open queuing network model, which is much simpler than the closed multi-class model we propose. (ii) It derives an analytical solution to a steady state version of the problem first and then targets this solution as a desired state through MPC with a quadratic trashing stage cost. In our case, this was not possible because we needed a sparse deployment at each step. A sparse deployment is one that most of services do not have replicas on most of the physical machines (i.e. the deployment matrix has many zeroes). For this reason, we had to use the sum of absolute values of the individual cost elements (as opposed to some of their squares).

5 Thesis Outline

The following structure is proposed as the structure of this thesis. Each item in the list corresponds to a chapter in the thesis:

1. **Introduction**. In chapter 1, we introduce the motivation and our approach. We motivate the three investigated problems in the thesis: Performance model estimation using dynamic clustering of user classes, optimal one-step-ahead resource share allocation for services, and optimal service replica placement via model predictive control.

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

3. **Related Work**. Chapter 3 describes the state of the art that is relevant to the introduced problems. Regarding the first problem (i.e. service demand estimation), the related work is classified into three categories: linear regression, nonlinear regression, and Bayesian estimation. We then provide a literature review of the second problem (i.e. service resource allocation with no configuration cost). Third, we provide a literature review for the service replica placement problem considering the trashing cost. Since this is a new research topic, the related work is quite limited.

4. **Models and State Estimation**. Chapter 4 presents the first contribution, improving the convergence of the estimator by dynamic group-

ing of the classes of service. We introduce the concept of modeling error and propose a dynamic clustering algorithm to improve the convergence. Then we prove the applicability of the approach through a set of real experiments. 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.

5. **Optimal Resource Share Adjustment in Cloud Using Dynamically Tuned Empirical Models**. Chapter 5 discusses our solution for the case that there is no trashing costs. It investigates the use of decomposed LQM and a single step ahead optimization. We first formulate the problem, introduce the modeling and estimation, and then introduce the optimization process. The proposed approach will be assessed by two cases studies of different scales.

6. **Optimal Service Replica Placement via Model Predictive Control**. In chapter 6, we target the OSP problem where reconfiguration cost is taken into account. Then we discuss the solution to the problem via Model Predictive Control framework. In this chapter, we first introduced the notation and defined the problem. We elaborate on the cost elements. There are a set of challenges in solving the introduced problems, including the non-linearity of the LQM, and the non-linearity of the infrastructure cost. We then discussed the fast (but not the exact) solution to the problem. The solution is transformed into a solver friendly format that can be handled by a common

convex optimization solver such as `cvx`.

We then explain the solution that considers the effect of the contention We show the behavior of the resulting optimization in a set of experiments using the FIFA98 workload and a synthetic service center. In a second set of simulations, we investigate the effect of lookahead horizon in the performance of the control. This will be done through several cost trade-off curves that compares the variation of the cost factors (i.e. the cost of SLA violations, the cost of resource, and the cost of trashing) four different look ahead horizons.

7. **Conclusion**. Chapter 7 concludes and discusses the possible future work.

References

[1] 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.

[2] 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.

[3] 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.

[4] Hamoun Ghanbari, Marin Litoiu, Przemyslaw Pawluk, and Cornel Barna. Replica placement in cloud through simple stochastic model predictive control. In *7th IEEE International Conference on Cloud Computing*, IEEE cloud'14, New York, NY, USA, 2014. ACM.

[5] 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.

[6] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, and Gabriel Iszlai. Feedback-based optimization of a private cloud. *Future Generation Computer Systems*, 28(1):104 – 111, 2012.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] T. Zheng, J. Yang, M. Woodside, M. Litoiu, and G. Iszlai. Tracking time-varying parameters in software systems with extended kalman fil-

ters. In *Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, page 345. IBM Press, 2005.

[22] 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.