Feasibility of Automated Service Management in Cloud Systems

Christopher C. Lamb, José Marcio Luna,

Pramod A. Jamkhedkar, Gregory L. Heileman,

Chaouki T. Abdallah

University of New Mexico

Department of Electrical and Computer Engineering

Albuquerque, NM 87131-0001

{cclamb,jmarcio,pramod54,heileman,chaouki}@ece.unm.edu

May 30, 2011

Abstract

In this paper, we examine the problem of a single provider offering multiple types of service level agreements, and the implications thereof. In doing so, we propose a simple model for machine-readable service level agreements (SLAs) and outline specifically how these machine-readable SLAs can be constructed and injected into cloud

infrastructures - important for next-generation cloud systems as well as customers. We then computationally characterize the problem, establishing the importance of both verification and solution, showing that in the general case injecting policies into cloud infrastructure is NP-Complete, though the problem can be made more tractable by further constraining SLA representations and using approximation techniques. Finally, we outline how to transform this problem into a more control-theoretic structure.

1 Introduction

The past few years have witnessed unprecedented expansion of commercial computing operations as the idea of cloud computing has become more mainstream and widely adopted by forward thinking technical organizational leadership. This rate of adoption promises to increase in the near future as well. With this expansion has come opportunity as well as risk, embodied by recent major service outages at leading cloud providers like Amazon. These issues promise to become more difficult to control as managed infrastructure expands. This expansion will simply not be possible without large amounts of automation in all aspects of cloud computing systems.

The current state of the art in cloud systems is poorly differentiated and not as customer-focused as it could be. Current providers place the responsibility of monitoring performance and proving outages on the consumer [1]. Furthermore, providers as a whole usually provide one type of service level

agreement (SLA) in a loosely-defined one-size-fits-all type of arrangement. This provides strong differentiating opportunities for smaller, second generation cloud system providers who have established the technology required to scalably manage multiple, competing SLAs on the same infrastructure in tandem with clear customer system visibility.

These second generation providers will rely on automated infrastructure management in order to scale. One of the first steps toward automating these systems is automating SLA management and compliance.

Herein, we will elaborate the idea of applying usage management to a single system governed by multiple different types of SLAs. We will define the problem, more formally describe SLAs, analyze the implications of that formality, and using this information design a prototypical control system.

In Section 2, this paper begins by describing the different types of cloud computing models that generally exist today and how they manage services. Immediately thereafter, we propose a possible future model in which users can have unique SLAs that more closely fit their needs rather than shoehorning their computing needs into a previously configured contract. Then, in Section 3, we more formally define an SLA, and show how to convert one to an evaluatable expression. In the following section, Section 4, we analyze the new SLA model and extract specific theoretical limits on computability and discuss implications thereof, showing SLAs in general to be NP-Complete. Finally in Section 5 we use our new conclusions to design a prototypical control system using these principles.

1.1 Previous Work

As cloud computing is emerging as the future of utility systems hosting for consumer-facing applications. In these kinds of systems, components, applications, and hardware are provided as utilities over the Internet with associated pricing schemes pegged by system demand. Users accept specific QoS guidelines that providers use to provision and eventually allocate resources. These guidelines become the basis over which providers charge for services.

Over the past few years multiple service-based paradigms like web services, cluster computing and grid computing have contributed to the development of what we now call cloud computing [2]. Cloud computing distinctly differentiates itself from other service-based computing paradigms via a collective set of distinguishing characteristics: market orientation, virtualization, dynamic provisioning of resources, and service composition via multiple service providers [3]. This implies that in cloud computing, a cloud-service consumer's data and applications reside inside that cloud provider's infrastructure for a finite amount of time. Partitions of this data can in fact be handled by multiple cloud services, and these partitions may be stored, processed and routed through geographically distributed cloud infrastructures. These activities occur within a cloud, giving the cloud consumer an impression of a single virtual system. These operational characteristics of cloud computing can raise concerns regarding the manner in which cloud consumer's data and applications are managed within a given cloud. Unlike other computing paradigms with a specific computing task focus, cloud

systems enable cloud consumers to host entire applications on the cloud (i.e. Software as a Service) or to compose services from different providers to build a single system. As consumers aggressively start exploiting these advantages to transition IT services to external utility computing systems, the manner in which data and applications are handled within those systems by various cloud services will become a matter of serious concern.

A growing body of research has begun to appear over the past two years applying control theory to tuning computer systems. These range from controlling network infrastructure [4] to controlling virtualized infrastructure and specific computer systems [5], [6] to exploring feedforward solutions based on predictive modeling [7]. Significant open questions remain to research within this field [8], [9].

2 Cloud System Models

Current cloud systems do not ignore SLA restrictions; rather, they are designed from the ground up to support a single type of SLA. That SLA generally encompasses total system uptime and some kind of response time metric [10, 11]. If for some reason the cloud provider can no longer adhere to the terms outlined, some kind of compensation strategy applies to affected customers. Future cloud providers can very well use the ability to support multiple SLAs as a way to differentiate available products from competitors.

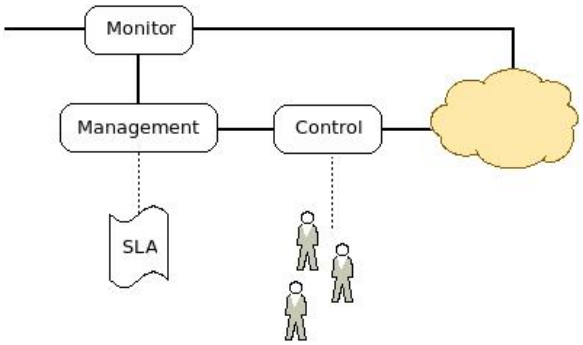

Figure 1: Single SLA with Control Elements

2.1 Current Model

Current systems like Amazon's EC2 or Rackspace products are designed around high availability, and this is reflected in the focus of their supplied SLAs. This common design focus is also evident in the artifacts generated by other vendors [12]. Furthermore, Amazon offers clear guidance on how to develop systems that take advantage of their robust architecture as well as services that provide some measure of automatic scaling [13, 14]. This combination of market leading position and products and the extensive supplied guidance make Amazon a clear choice to examine when reflecting on the current state-of-the-art.

Amazon's Cloud Watch products used in tandem with Auto Scaling provide the ability to control the number of deployed instances in response to specific system loads, as shown in Figure 1 [1, 15]. Cloud Watch gives customers the ability to monitor various system performance metrics for their virtual machines, including but not limited to latency, processor use, and re-

quest counts. Furthermore, users can set resource levels at which additional EC2 instances are created or destroyed. This provides some level of personalized management and control over deployed systems within Amazon's cloud infrastructure.

2.2 Future Reference Model

While current cloud service providers focus on a single quality-of-service metric, future providers may very well begin to provide multiple metrics over which they will define service levels, as shown in Figure 2. This is not without precedent - just as airlines provide the same essential product at different service points, cloud providers could supply system hosting via disparate service levels, including divergent service metric definitions. For example, current architectures support uptime and availability as the primary managed metric from an SLA perspective. Future architectures could support uptime and availability, as well as specific latency, bandwidth, and geo-location sensitive hosting parameters. These kinds of SLAs would also continue to outline penalties when any of the conditions of that SLA were violated. Unlike current SLAs however, these could also differentiate based on the magnitude of the imposed penalty, with different classifications of service mapping to increasingly large penalties on service failure.

While industry does seem to certainly be trending in this direction, as indicated by the development of tools supporting user-centric infrastructure monitoring and management, this kind of control is not yet embedded into

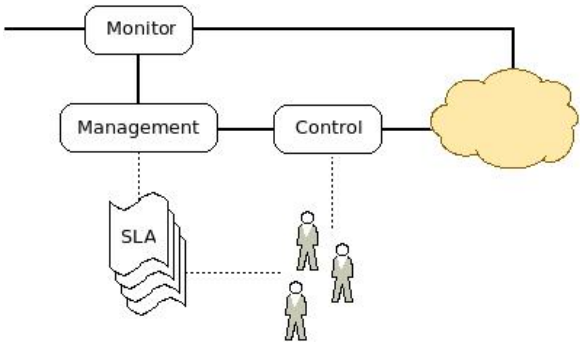

Figure 2: Multiple SLA Architectural Integration

contracts of any kind, much less agreements that are machine-readable. Furthermore, this kind of management is still manual and cannot scale to the levels needed to manage Internet-scale systems.

3 Service Level Agreements Defined

As we have seen, SLAs generally consist of a set of conditions of use under which the SLA is binding, a set of obligations that the provider will adhere to if the customer adheres to the set conditions, and two sets of penalties, one penalizing the provider when breaching obligations, and another penalizing the customer when breaching conditions of use. Conditions are generally loosely defined, while provider obligations are much more rigorously constructed. Generally however, conditions and obligations in this context can be viewed as defined by *objectives* which are measured by *indicators*. In the case of provider-centric obligations, these are commonly defined as Service Level Indicators (SLIs) and Objectives (SLOs).

With this general understanding of SLAs and related SLIs and SLOs, we can create a non-specific definition of an SLA as a set of tuples:

$$SLA = \{(I, O, E, P)_{0..n}\}, n \in Z \tag{1}$$

Where I is an indicator function, $\forall i \in I, i : () \to \tau$, which retrieve indicator values, generally related to some kind of SLI or customer condition indicator. O is a set of values derived from SLOs or customer condition objectives such that $\forall o \in O, o : P(\tau)$. E is a set of predicates that indicate whether a specific indicator complies with it's objectives, where $\forall e \in E, e : (() \to \tau) \times P(\tau) \to bool$. P is a set of penalty functions, $\forall p \in P, p : Z \to Z$, where the first argument is generally an elapsed time value.

For example, say we are a customer of Nimbus Cloud Corporation, and we have an SLA in which Nimbus provides guaranteed 100% uptime and packet latency between 300 and 650 milliseconds. Nimbus does not have any customer conditions specified within it's SLAs. This then gives us a machine

evaluateable SLA:

$$
\begin{aligned}
SLA_{nimbus} = ((&uptime\_monitor() : bool, \\
&\{true\}, \\
&uptime\_evaluator(monitor : () \rightarrow bool, \{true\}) : bool, \\
&uptime\_penalty\_evaluator(T : Z) : Z), \\
&(latency\_monitor() : Z, \\
&\{300, 650\}, \\
&latency\_evaluator(monitor : () \rightarrow Z, \{300, 650\}) : bool, \\
&latency\_penalty\_evaluator(T : Z) : Z))
\end{aligned}
\tag{2}
$$

This more rigorous SLA allows users to monitor obligations and determine penalties when triggered.

4 Controlling with Service Level Agreements

Now that we have more rigorously defined our SLAs, notice that the SLA evaluation functions are predicates, and can be curried for later execution if needed. This allows us to begin a more fundamental analysis of SLAs and their capabilities.

4.1 Computational and Space Complexity

In Section 3, we defined an SLA to essentially be a sequence of evaluatable predicates. These evaluatable predicates are related in some way; currently, an SLA is the conjunction of these predicates. As these predicates can be created prior to evaluation and at evaluation time require no specific arguments once appropriately curried, we can define these predicates as as boolean *terms*. Ergo, once we've created a group of predicates and transformed them into terms, we are evaluating an arbitrary boolean equation - in other words, we are verifying an instance of the Boolean Satisfiability Problem, or SAT.

SAT is NP-Complete, and very difficult to solve on today's computing systems [16]. $3SAT$, a subset of SAT, is equally difficult, while $2SAT$ is not. $2SAT$ is firmly in the computational class P; in fact, $2SAT$ is NL-Complete as well, so we know it is solvable in an amount of space logarithmic in the number of boolean terms[17]; it is widely held that both SAT and $3SAT$ cannot be solved in logarithmic or less space. Finally, as $2SAT$ is NL-Complete, we also know it is contained within NC^2, and as such is highly parallelizeable [17].

4.2 Verification v. Solution

General SLA use focuses on verification rather than solution. That is to say, with respect to a given SLA, both the user and provider is more concerned with whether the system is currently compliant with all SLA terms. In future

use, this may very well no longer be the case. For example, imagine a cloud system from the user's perspective that spans multiple could providers; a single general purpose provider for general computing, data storage, and queuing, a specific-use provider for an unique set of algorithms of some kind (say, market modeling algorithms), and finally a Content Delivery Network (CDN) provider. Each of these providers have a unique SLA with multiple conditions. In this particular case, the user may need to know if the given system can work together at all under the terms of the SLAs, and if so, under what conditions. As the user has combined all the SLAs composing the system and is attempting to find some combination of terms that satisfies the resulting boolean formula, the user is in essence attempting to find a solution for this instance of SAT, a known NP-Complete problem. Likewise, a cloud provider may need to solve similar problems where the SLAs at issue are the whole of SLAs issued to the entire provider's customer base.

Nontrivial generalized SLAs may be too difficult to solve effectively without using some kind of approximate heuristic or SAT Solver [18, 19]. If these SLAs are formulated in at most a $2SAT$ style problem however, they are suddenly much more tractable, easier to work with, and amenable to efficient solution. Keep in mind, even generalized SLAs can be efficiently verified.

5 Control Theoretic Model

Modeling the behavior of the workload in a computing infrastructure is challenging given the difficulties to non-linear identification techniques, the model uncertainty generated by the complexity of the system as well as the Multi-Input Multi-Output (MIMO) nature of the system. Several alternatives have been presented to cover the main features of the workload in a computing infrastructure such as the fluid approximation model [20], or the approximation of the relation between the resource allocation and a given performance measure [21],[22].

Optimal control is a common approach to calculate the resource allocation according to the actual workflow and the power consumption of the system by minimizing a cost function. This minimization can be carried out by using partial derivatives [23],[20], Linear Quadratic Control (LQC) [24],[21], or by applying Model Predictive Control (MPC) [22],[25]. The optimization problem consists of calculating a control input such that the cost of the control law is minimized while the required SLO of a set of applications is partially or completely reached.

5.1 Model Estimation and Optimal Control

In previous works [23],[21], the implemented control technique is composed of an online model estimator and a Multi-Input Multi-Output resource allocation controller as shown in Fig. 3. The main goal of this controller is

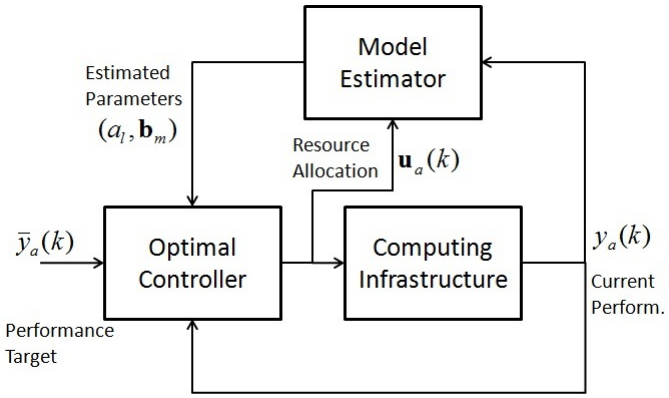

Figure 3: Block diagram of the Optimal Controller with Model Estimator.

to optimally allocate the virtualized resources on a computing infrastructure so that the system can fulfill the required SLO. In current data center architectures, every tier of an application resides in a *virtual machine* (VM) and a multi-tier application can run on several nodes. In this set up, several applications can be executed in one node, thus reducing the number of nodes in the computing infrastructure and minimizing costs.

The model estimator uses an Auto-Regressive (AR) model to approximate the relation between the resource allocation of the application $a \in A$, namely $\mathbf{u}_a(k) = \{\mathbf{u}_{a,r,t} : r \in R, t \in T_a\}$ and the peformance output $y_a(k)$ related to a. The AR model proposed in [23] is given by,

$$\begin{aligned} y_a(k+1) &= \sum_{l=1}^{n} a_l(k) y_a(k+1-l) \\ &+ \sum_{m=0}^{n-1} \mathbf{b_m}^T(k) \mathbf{u}_a(k-m). \end{aligned}$$

We denote by $\mathbf{u}_{a,r,t}$ the actual allocation of the resource $r \in R$ for the application $a \in A$ in tier $t \in T_a$. A is the set of all hosted applications, R is the set of all resource types considered and T_a is the set of all tiers associated to application a. The scalar parameters $a_l(k)$ and the parameter vectors $\mathbf{b_m}(k)$ are estimated through a recursive least square (RLS) algorithm. The optimizer searches for the required allocation vector $\bar{\mathbf{u}}_a$ that guarantees the performance target of the application a while avoiding oscillations in resource allocation. This may be achieved by calculating $\bar{\mathbf{u}}_a$ that minimizes the following cost function,

$$J_a = \left(y_a(k+1) - \bar{y}_a(k+1)\right)^2 + q\|\bar{\mathbf{u}}_a(k) - \mathbf{u}_a(k-1)\|^2. \tag{3}$$

where $\bar{y}_a(k)$ is the desired performance associated to the application a. Note that the function J_a is minimized for $\left(y_a(k) - \bar{y}_a(k)\right)^2 \approx 0$ leading the application a to approach the desired performance. Furthermore (3) is minimized if the difference $\|\mathbf{u}^*{}_a - \mathbf{u}_a(k-1)\|$ becomes smaller *i.e.* when large changes in the resource allocation during a sample period are avoided. By solving $\frac{\partial J_a}{\partial \mathbf{u}(k)} = 0$ for $\mathbf{u}_a(k)$ the optimal control law for resource allocation is given by,

$$\begin{aligned}\bar{\mathbf{u}}^*_a(k) &= (\mathbf{b}_0\mathbf{b}_0{}^T + qI)^{-1}\left[\left(\bar{y}_a - \sum_{l=1}^{n} a_l y_a(k-l+1)\right.\right. \\ &\quad \left.\left. - \sum_{m=1}^{n-1} \mathbf{b}_m^T \mathbf{u}_a(k-m)\right) + q\mathbf{u}_a(k-1)\right],\end{aligned}$$

with I the identity matrix and $q \in \Re$ a scaling factor. Note that, a_i, b_j with $i = 1, 2, \cdots, n$, $j = 0, 1, 2, \cdots, n-1$ are functions of k.

It might happen that the computing infrastructure cannot satisfy the resource allocation that has been calculated by the controller. On that matter, Padala *et. al.,* present in [21] a complementary layer in the workload control called *NodeController*. This module takes the resource allocation calculated by the optimal control (called *AppController* in [21]) and, if the total requested allocation is less than the available capacity of the system, it assigns the resources as the optimal controller dictates. On the other hand, when the total resource allocation is higher than the available capacity we have *contention*. In such cases, a new optimization problem is carried out in order to minimize the error between the required application performance and the actual one.

Padala *et. al.,* propose the following example in [21]. Two nodes namely, n_1 and n_2 hosting applications 1 and 2, sharing CPU and disk. Then, from application 1 we get the requests $\bar{u}_{1,cpu,web}$ and $\bar{u}_{1,disk,web}$. Similarly, from application 2 we get the requests $\bar{u}_{2,cpu}$ and $\bar{u}_{2,disk}$. Let us assume that node n_1 has enough capacity to fulfill the request for only one of the available resources. Modeling the shortage of resources we get $\Delta u_{1,r,web} = \bar{u}_{1,r,web} - u_{1,r,web}$ and $\Delta u_{2,r} = \bar{u}_{2,r} - u_{2,r}$. This problem is expressed in [21] as a constrained opti-

mization problem,

$$\begin{aligned} \min J_{n1} &= w_1 \left(\frac{\partial y_1}{\partial u_{1,r,web}} \Delta u_{1,r,web} \right)^2 \\ &\quad + w_2 \left(\frac{\partial y_2}{\partial u_{2,r}} \Delta u_{2,r} \right)^2 \end{aligned} \tag{4}$$

subject to

$$\Delta u_{1,r,web} + \Delta u_{2,r} \geq \bar{u}_{1,r,web} + \bar{u}_{1,r,web} + \bar{u}_{2,r} - 1 \tag{5}$$

$$\Delta u_{1,r,web} \leq 0, \tag{6}$$

$$\Delta u_{2,r} \leq 0. \tag{7}$$

where $\frac{\partial y_1}{\partial u_{1,r,web}} \Delta u_{1,r,web}$ estimates the error between the requested and actual performance of application 1. The condition in (5) imposes a capacity constraint that when violated means contention of one of the resources. The conditions given by (6) and (7) guarantee that no application can exceed its target performance to the detriment of the other. The weights w_1 and w_2 determine the priority of the applications and the lower the weight, the greater the degradation.

Getting back to our example of Nimbus Cloud Corporation, we can map the SLAs to our actual control problem. Notice that the input of the system in Fig. 3, corresponds to the vector of required performance measurements for each application $a \in A$, then each entry of the vector is $\bar{y}_a(k)$. Let us assume that the outputs of the system are *uptime* t_u, *latency* ℓ and *abandon*

rate α, which are measurable. Latency is the time a server needs to process a request from the client. Abandon rate of a server is the ratio between the rejected requests and the total number of received requests. Nimbus Cloud Corporation is able to assign the SLO values per application we require as a client and put them in the input array $\bar{y}_a(k)$. Since the values are fixed thresholds of performance, the indicator functions of performance are straighforward, *e.g.,*

$$\left(\ell \geq 300ms \wedge \ell^{-1} \geq \tfrac{1}{650ms}\right)$$

$$(t_u \geq 95\%)$$

$$\left(\alpha^{-1} \geq \tfrac{1}{10\%}\right)$$

Using the previous conditions the indicator functions will return *true* if $300ms \leq \ell \leq 650ms$, similarly it will return *true* if the minimum uptime is provided and the abandon rate is under the indicated threshold. Notice that other performance indexes can be used such as CPU utilization, memory usage or even the number of active processors in the computing infrastructure. Then, the SLO and actually the Service Level Indicators (SLI) are easily mapped into the depicted theoretical controls approach.

6 Conclusions and Future Works

Herein, we started by going over the current generalized state of most cloud systems from an SLA perspective, differentiating between current architectures that incorporate SLA ideas into the design itself with possible futures architectures that incorporate pluggable SLAs with varying indicators and objectives. We then generalized SLAs into sets of quadruples containing a monitoring function, a set of values defining acceptable ranges returned from the monitoring functions, an evaluation function, and a penalty evaluating function, and demonstrated how this formulation could be used with a specific example. With this in place, we then demonstrated that the generalized SLA problem is equivalent to SAT, and therefore is NP-Complete. We finally covered the implications and theoretical limits implied by this NP-Completeness, validating the applicability of this work by designing a realistic control model using these ideas.

This is preliminary work into establishing the theoretical bounds surrounding effective automated control of cloud systems within Internet-scale systems. Furthermore, the SLA modeled was fairly simple, and only took into account externally-evaluatable metrics in a black-box arrangement. SLAs can very well outline other operational parameters, like specific data routing, fine-grained usage management, or encryption requirements. These scenarios are much more difficult to manage than the kinds outlined within this paper. Likewise, experimental evidence supporting these control ideas will be vital

to promoting acceptance and action around these concepts within both cloud service provider systems and user configured applications.

References

[1] "Amazon Cloudwatch," http://aws.amazon.com/cloudwatch/, May 2011.

[2] R. Buyya, "Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2009.97

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[4] Y. Ariba, F. Gouaisbaut, and Y. Labit, "Feedback control for router management and tcp/ip network stability," *Network and Service Management, IEEE Transactions on*, vol. 6, no. 4, pp. 255 –266, 2009.

[5] Z. Wang, Y. Chen, D. Gmach, S. Singhal, B. Watson, W. Rivera, X. Zhu, and C. Hyser, "Appraise: application-level performance management

in virtualized server environments," *Network and Service Management, IEEE Transactions on*, vol. 6, no. 4, pp. 240 –254, 2009.

[6] M. Kjaer, M. Kihl, and A. Robertsson, "Resource allocation and disturbance rejection in web servers using slas and virtualized servers," *Network and Service Management, IEEE Transactions on*, vol. 6, no. 4, pp. 226 –239, 2009.

[7] S. Abdelwahed, J. Bai, R. Su, and N. Kandasamy, "On the application of predictive control techniques for adaptive performance management of computing systems," *Network and Service Management, IEEE Transactions on*, vol. 6, no. 4, pp. 212 –225, 2009.

[8] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin, "What does control theory bring to systems research?" *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 62–69, January 2009. [Online]. Available: http://doi.acm.org/10.1145/1496909.1496922

[9] J. Hellerstein, S. Singhal, and Q. Wang, "Research challenges in control engineering of computing systems," *Network and Service Management, IEEE Transactions on*, vol. 6, no. 4, pp. 206 –211, 2009.

[10] "Amazon EC2 SLA," http://aws.amazon.com/ec2-sla/, May 2011.

[11] "Rackspace Cloud - SLA," http://www.rackspace.com/cloud/legal/sla/, May 2011.

[12] J. Dean, "Designs, Lessons and Advice from Building Large Distributed Systems." Presented at the Large-Scale Distributed Systems and Middleware (LADIS) Conference, 2009.

[13] J. Varia, "Architecting for the Cloud: Best Practices," January 2010. [Online]. Available: http://aws.amazon.com/architecture/

[14] J. Barr, A. Narin, and J. Varia, "Building Fault-Tolerant Applications on AWS," May 2010. [Online]. Available: http://aws.amazon.com/architecture/

[15] "Auto Scaling," http://aws.amazon.com/autoscaling/, May 2011.

[16] M. Sipser, *Introduction to the Theory of Computation*, 1st ed. International Thomson Publishing, 1997.

[17] C. M. Papadimitriou, *Computational complexity*. Reading, Massachusetts: Addison-Wesley, 1994.

[18] D. S. Hochbaum, Ed., *Approximation algorithms for NP-hard problems*. Boston, MA, USA: PWS Publishing Co., 1997.

[19] "Sat competitions," http://www.satcompetition.org/, May 2011.

[20] L. Malrait, "Qos-oriented control of server systems," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 59–64, July 2010.

[21] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized re-

sources," in *Proceedings of the 4th ACM European conference on Computer systems*. Nuremberg, Germany: ACM, March 2009, pp. 13–26.

[22] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for qos-aware clouds," in *Proceedings of the ACM European Society in Systems Conference 2010*, Paris, France, April 2010, pp. 237–250.

[23] T. Abdelzaher, Y. Diao, J. L. Hellerstein, C. Lu, and X. Zhu, "Introduction to control theory and its application to computing systems," in *SIGMETRICS Tutorial*, Annapolis, MD, June 2008, pp. 185 – 215.

[24] Y. Fu, C. Lu, and H. Wang, "Control-theoretic thermal balancing for clusters," in *International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID'09)*, April 2009, invited paper.

[25] Z. Wang, N. Tolia, and C. Bash, "Opportunities and challenges to unify workload, power and cooling management in data centers," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 41–46, July 2010.