Gradient-based Models of Multitier Systems

Shuyi Chen[1], Kaustubh R. Joshi[2], Matti A. Hiltunen[2],
Richard D. Schlichting[2], and William H. Sanders[1]

[1]Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL, USA
{schen38,whs}@illinois.edu

[2]AT&T Labs Research
180 Park Ave.
Florham Park, NJ, USA
{kaustubh,hiltunen,rick}@research.att.com

I. Introduction

Predicting the impact of environmental and infrastructure changes on end-to-end performance metrics of multitier systems is an important endeavour with tremendous opportunities. Multitier applications shape a large portion of the online experience by powering services such as e-commerce portals, search engines, social-networking sites, collaborative services, email, and enterprise management systems. Studies such as [1], [2], [3] and the experiences of large service providers [4] have repeatedly shown the importance of performance, measured using metrics such as end-to-end response time, on user satisfaction, traffic growth, and, consequently, business viability. Other metrics such as a service's overall power consumption are crucial for energy saving and green computing efforts. Predictive models for such metrics can aid in a variety of system management tasks such as system deployment, planned upgrades, and adaptation in response to dynamic workload variations.

However, the design and deployment life-cycle of typical online applications poses significant practical challenges that make the use of traditional modeling techniques such as queuing theory difficult. Often, there is simply not enough knowledge about the application or the infrastructure it is deployed on to construct detailed models. Increasing software complexity and the use of poorly understood off-the-shelf components as "black-box" subsystems exacerbate the problem. Application deployment environments also play a role. As multitier systems are increasingly deployed in outsourced data centers, shared infrastructure clouds, content distribution networks, and over complex networks, the application owners are often separate entities from the infrastructure providers. The former do not have enough visibility into the infrastructure to construct good models, while the later do not have visibility into the application to do so. Finally, even if detailed information were available, models often require significant expertise and entail significant expense, both of which are scarce in commercial environments with intense budget and time-to-market pressures.

In this paper, we propose an alternative approach to detailed performance modeling of multitier systems based on simple models constructed via automatic on-line computation of "resource gradients". For a multitier application, resource gradients quantify the impact of changes in low level system resource parameters (or "knobs") such as individual node CPU capacity, CPU frequency, or network link latency on the end-to-end performance metrics of the system by computing local point derivatives of the metrics with respect to the parameters. This is done by injecting minute perturbations into a running system and using measurements of its response to estimate the derivatives. Resource gradients provide models that capture the impact of application specific parameters and infrastructure configurations, but yet, require very little information about the target system, it's structure, resource usage characteristics, and communication patterns. In Section II, we define resource gradients and describe how they can be measured even in noisy production environments with very low perturbation using spectral analysis techniques based on Fourier transforms.

Gradient-based models can be very accurate for small parameter variations. However, in return for their simplicity and ability to model black-box systems, they increasingly diverge from reality as the environment changes become more significant compared to the original measurement point, and thus must be periodically recalibrated. The divergence is due to the linearity assumptions made by the models that real systems often do not abide by for their whole range of possible parameter values. While it is not possible to completely eliminate the divergence, we have developed techniques to deal with many types of non-linear behavior and extend the useful range of resource gradient models. We demonstrate these techniques in the context of different applications (i.e., different metric and knob combinations) in Section III and show using experimental results that gradients provide a practical and powerful modeling paradigm for a diverse range of applications ranging from topology aware system deployment to virtual machine consolidation to energy conservation.

II. Technical Approach

We begin by defining resource gradients, and then describe how they can be measured in a push-button manner.

A. Resource Gradients

Consider a multitier application consisting of a set of nodes represented by vector $N = (n_1, n_2, \ldots, n_m)$ and connected by a set of logical communication links represented by vector $L = (l_1, l_2, \ldots, l_n)$. Each node represents a single software component of a specific type, e.g., a web server, application server. Nodes execute using resources (e.g., physical hosts) that may be dedicated or shared. E.g., several nodes may execute within virtual machines on the same physical host. Logical links exist between two nodes whenever message exchanges occur between them during system operation. Each logical link may comprise of many physical network links. Together, nodes and links are called the *elements* of the application. Each element e is associated with a vector of attributes $A = (a_1^e, a_2^e, \ldots, a_k^e)$ that quantify properties of the node or its resources that may impact its performance. For example, attributes may include the capacity of a resource available to an element such as the fraction of the host's CPU or I/O bandwidth available to a node, or the bandwidth available to a logical link. They can also include properties of the resources themselves, e.g., the CPU speed of the host a node runs on, or the link latency of a logical link.

Applications are also associated with an end-to-end "progress metric" M. This progress metric may include any single measurable end-to-end property important to the application, and may be different for different kinds of applications. For example, it could be the execution time in case of non-interactive applications. For batch-oriented data processing applications, it could be the rate at which the application consumes input or produces output in terms of records per second. For transactional systems, the metric might be the throughput or the end-to-end response time of different types of application transactions. It can also quantify non-performance related attributes such as total power usage of the system or power consumption per transaction type. In this paper, we consider transactional systems whose users interact with them through a set of transactions, such as "login", "buy", and "browse", each of which utilizes a set of components according to a transaction-specific call graph. We focus on the end-to-end transactional response time of such transactions.

The goal of a resource gradient model is to quantify the relationship between attributes and end-to-end progress metrics. Specifically, consider the values of a single type of attribute A_k e.g., latency, for all p elements of the system in the current operating configuration c_0, i.e., $A_k(c_0) = (a_k^{e_1}(c_0), a_k^{e_2}(c_0), \ldots, a_k^{e_p}(c_0))$. We represent the relationship of the attribute to the metric M as an unknown function at the current operating point, or, $M = F(A_k(c_0))$. Then, the question we wish to answer is, "Given the value of $M(A_k(c_0))$ at the system's current operating configuration c_0, what is its value at a different operating configuration c_1, i.e., $M(A_k(c_1))$?".

To answer this question, the approach we take is as follows. Let the vector $\Delta A_k = A_k(c_1) - A_k(c_0) = (\Delta a_k^{e_1}, \ldots, \Delta a_k^{e_p})$ be the differential change in the attribute values between the current and the new operating configurations. Assuming that the function F is differentiable, we can then use the Taylor expansion to represent the desired $M(A_k(c_1))$ as:

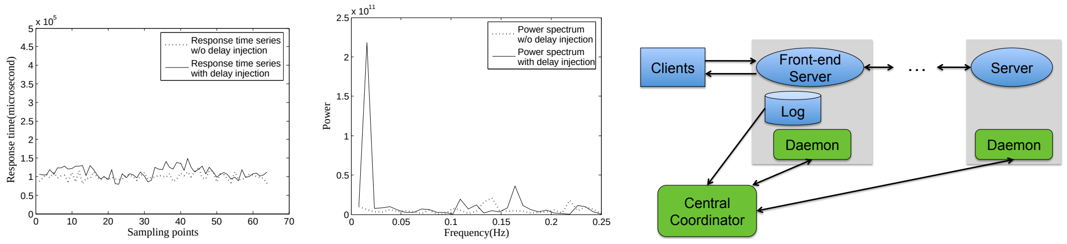
$$M(A_k(c_1)) = M(A_k(c_0)) + \sum_{e \in N \cup L} \left. \frac{\partial F}{\partial a_k^e} \right|_{c_0} \Delta a_k^e + O(\Delta A_k^2) \quad (1)$$

In this equation, the $O(\ldots)$ term represents the higher order derivatives and powers of the attribute values. If the derivatives $\frac{\partial F}{\partial A_k}\big|_{c_0} = (\frac{\partial F}{\partial a_k^{e1}}\big|_{c_0}, \frac{\partial F}{\partial a_k^{e2}}\big|_{c_0}, \ldots)$ (i.e., the gradients) are known at the current operating configuration c_0, one might imagine using this equation to predict the performance of the system in the new configuration by ignoring the higher-order derivatives and powers in $O(\Delta A_k^2)$. However, doing so is justifiable only if ΔA_k is small enough to cause the higher powers to vanish, or if F is linear, thus ensuring that the higher-order derivatives are zero. In practice, changes in the operating conditions could be large, making the first condition impractical. The second condition can hold true depending on the type of metric and attribute being considered. In Section III, we discuss one type of gradient—the link gradient—for which that is the case. However, in general, such linearity assumptions may not hold, and non-linearity can impact the accuracy of the gradient. Although non-linearity can always be overcome by recalculating the gradients whenever they change, it is important to minimize the need for such recalculations not only to reduce runtime measurement overhead, but also to prove a meaningful operating range over which gradient models can make accurate predictions.

To tackle the problem of non-linearity of the unknown function F with respect to the attributes A_k, we recast F in terms of "basis functions" $B_k = (b_k^{e_1}, \ldots, b_k^{e_p})$ with respect to which it is linear. For each element e, the basis function $b_k^e(A_k(c_1))$ is a function whose values for a configuration c_1 can be computed solely based on the values of attributes in that configuration, i.e., $A_k(c_1)$ and any constant parameters. As we show in Section III, in many cases, these basis functions can be derived using high level knowledge of the causes of non-linearity without the need for detailed application knowledge. Since the value of basis functions can be predicted for a new configuration, the change in basis functions between the old and new configurations, i.e., $\Delta B_k = |B_k(A_k(c_1)) - B_k(A_k(c_0))| = (\Delta b_k^{e_1}, \ldots, \Delta b_k^{e_p})$ can be used along with a gradient with respect to the basis functions, i.e., $\frac{\partial F}{\partial B_k}$ to predict the value of the metric in a new configuration, or $M(A_k(c_1)) \approx M(A_k(c_0)) + \frac{\partial F}{\partial B_k} \cdot \Delta B_k$ without any error.

B. Spectral Gradient Measurement

The basic technique we use to estimate the resource gradient for each system element e with respect to attribute a_k^e at runtime is conceptually very simple. We inject small perturbations in the value of a_k^e, and then measure the corresponding change in the end-to-end metric. The ratio of the change in the metric to the change in the basis function value provides the gradient. However, the problem with such an approach is that measurements made on running systems are often very noisy, especially when resources are shared. Therefore, to get accurate estimates with tight confidence intervals, measurements must either be accumulated over long periods of time, or the perturbation must be high enough to overcome the effects of noise. Both approaches are less than ideal. Conducting measurements over long time intervals carries the risk that the system behavior might change during measurement as a



(a) Response Time Series (b) Frequency Spectrum (c) Gradient Computation Architecture

Fig. 1. Gradient Measurement

result of workload changes, or other runtime adaptations. On the other hand, injecting large perturbations can be intrusive and be detrimental to user experience.

To address this problem, we have developed a technique that uses the observation that while noise may be present in a system's runtime measurements, it is rarely periodic. While some sources of noise such as garbage collection may indeed be periodic, such sources are few, and operate at a small set of frequencies that can be easily identified by examining the metric in the frequency domain. The remainder of the noise is often spread uniformly across many different frequencies, with very little contribution at any particular frequency. Therefore, if the perturbation is crafted such that a large portion of its energy is concentrated on a single frequency at which noise is usually low, one can get significantly superior signal-to-noise ratios and more accurate measurements.

For simplicity, we use a square wave pattern for injecting perturbations. Within a short time frame (usually several minutes), we perturb the resource attribute at a single element by repeatedly switching its value between its normal value and a high/low value at a frequency chosen to minimize ambient noise. This also causes a square-wave perturbation in the basis function. Subsequently, we use standard Fourier transforms to compute the frequency spectrum of the time series of metric measurements made during the period of signal injection, and use that frequency spectrum to estimate the resource gradient using the following equation, as derived in [5].

$$\frac{\partial M}{\partial b_k^e} = \frac{|\mathrm{FFT}^d(M) - \mathrm{FFT}^0(M)| \cdot sin(\frac{\pi}{2n})}{\Delta b_k^e \cdot f_d} \tag{2}$$

In this equation, n is the number of sample measurements in the metric time series, f_d represents the frequency at which the perturbation was injected, and $\mathrm{FFT}^d(M)$ and $\mathrm{FFT}^0(M)$ represent the Fast Fourier Transform (FFT) of the metric time series with and without the perturbation, respectively. Sample end-to-end response time series with and without perturbation along with their frequency domain counterparts are shown in Figures 1 (a) and (b). As can be seen, even a square wave that is visually difficult to discern in the time domain results in large spikes in the frequency domain. Use of this technique allows a reduction of noise and perturbation by an order of magnitude (see [5]) compared to a time-domain approach, and thus makes it possible to recalculate the gradients dynamically and cheaply while the system is running.

To facilitate gradient model computation, we have implemented a distributed active monitoring framework, as shown in Figure 1(c), that automatically calculates the frequency gradients for a distributed application. The framework consists of a central coordinator and a set of local daemons on each node. Each daemon is responsible for reporting the current values of node and link attributes to the central co-ordinator, and changing their values on command from the coordinator. Depending on the attribute being changed, several different perturbation mechanisms are used as described in Section III. The central coordinator orchestrates the daemons and executes the gradient measurement algorithm. It requires a list of the machines executing the application's components and the location of the end-to-end metric data. No additional workload beyond the application's normal workload is required for measurement purposes, thus ensuring minimal interference in a running system.

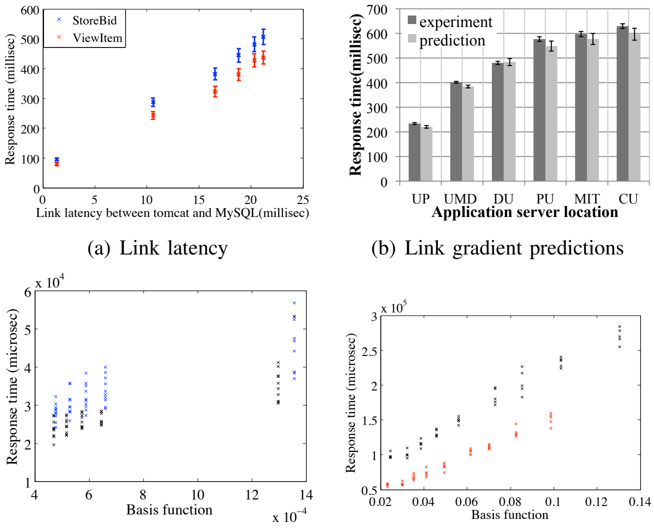
The process of measuring the gradients consists of two phases: a) the training phase, and b) a set of per-element measurement phases. In the training phase, the coordinator passively collects the metric values, and uses them to determine the parameters for the perturbation square wave that is to be injected into each individual element. The per-element measurement phase is conducted once for every element, and is the active phase during which perturbations are actually introduced. A detailed description of the algorithms can be found in [5] and [6].

III. Applications of Gradients

Next, we describe three different gradient models that we have developed, and experimentally show that they are able to achieve sufficient linearity and accuracy over a wide range of parameters without retraining. Our experimental setup consists of a deployment of RUBiS, which is an eBay-like auction system widely used as a benchmark in the literature. We use a 3-tier Java servlet version of the application with a front-end Apache web server, a middle-tier Tomcat application server, and a back-end MySQL database server. The application is subjected to a workload generated by the standard client simulator program provided with the application.

A. Link Gradients

The first example of resource gradients that we demonstrate are *link gradients*. They are defined as the rate of change of the system's end-to-end transaction response time as a function of changes in the link latency of its logical links. During link gradient estimation, end-to-end response times



(a) Link latency

(b) Link gradient predictions

(c) Physical CPU frequency

(d) Virtual CPU frequency

Fig. 2. Gradients

are measured by parsing the front-end web server access logs. Perturbations to link latency are injected by redirecting packet-level traffic on the application's communication links to the gradient measurement local daemons where it is delayed for a constant amount of time while the perturbation signal is high. Redirection is accomplished through per-node kernel level mechanisms such as the Linux *iptables* packet filter.

We choose an identity basis function for link gradients. The reason is that the relationship between end-to-end response time and a link's latency is based primarily on the communications pattern between the two nodes that communicate over the link, i.e., whether the communication is synchronous, asynchronous, or pipelined, or if there is a cache or load balancing involved. Since these communication patterns are often relatively constant for a given application, the relationship between link latency and end-to-end response time is usually linear. This is also experimentally demonstrated by Figure 2(a) which shows the linearity of the end-to-end response time of two randomly chosen transactions as a function of the link latency between the application and database server at a workload of 40 req/sec. Large scale experiments [5] on PlanetLab nodes distributed across the US have further corroborated the linearity for latency changes at intra-continental scales.

Figure 2(b) shows the accuracy of the end-to-end response time predictions made by the link gradient models when the middle tier application server for the RUBiS test instance is moved across several widely distributed locations spanning the eastern coast of the United States[1]. As can be seen, the link gradients, which were calculated just once on our local testbed, can provide excellent predictions even in configurations that are dramatically different from the measurement setup. Link gradients find wide application in optimizing the deployment of distributed applications across multiple geographically dispersed sites. For example, they can be used to predict the impacts of moving parts of an application from one data center to another, of moving parts of an application into the cloud, of using a cache-based content distribution network in an attempt to improve responsiveness, or of optimizing the placement of shared back-end databases used by multiple applications.

B. Frequency Gradients

The next example of gradients is the *frequency gradient* which is defined as the rate of change of a system's end-to-end transactional response time with respect to changes in the CPU frequencies f_n of individual nodes. Such gradients can not only be used to predict the impact of upgrading machines, but can also be used to perform dynamic energy saving by slowing down nodes to the limits allowed by their response time SLAs. In this case, perturbations are injected by using dynamic voltage and frequency scaling (DVFS) features found in most modern microprocessors. The Linux *CPUfreq* system interface enabled by the *userspace* CPU frequency scaling governor is used to change CPU frequency.

Unlike link gradients that are substantially linear, CPU gradients demonstrate significant non-linearity. This is because frequency changes impact queuing behavior in significant ways, and cause a superlinear increase in end-to-end response times. Recognizing queuing as the primary source of non-linearity, we use a basis function based on the mean response time relation for a single M/G/1/PS queue, i.e., $rt = \frac{s}{1-U}$, where s is the per-transaction service time and U is the utilization. Subsequently, we utilize the fact that service time and utilization are inversely proportional to frequency f_n, while utilization is also proportional to the application workload w, i.e., $s \propto 1/f_n$, and $U = \alpha_n w/f$, where α_n is a node specific constant. Therefore, we set the basis function for a node n to $b_a^n = \frac{1}{f_n - \alpha_n w}$. Note that although the basis function is based on the high-level knowledge that non-linearity is caused by queuing effects, no application specific information such as service times or routing matrices are required. The only application specific constant is α_n, which can be characterized as the gradient for node utilization with respect to the basis function w/f_n. The estimation techniques described earlier can be used to compute both this utilization gradient and subsequently the frequency gradient.

While we do not provide a comparison of the gradient predictions with experimental measurements due to a lack of space, Figure 2(c) validates the linearity of the metric with respect to the basis function by plotting the system's mean end-to-end response times against the values of the basis functions as the CPU frequency for the Tomcat server is changed for two different workloads of 40 and 50 requests per second. The figure was obtained using a heterogeneous testbed in which the Apache server was hosted on an Intel E8400 Core 2 Duo machine with 4 operating frequency choices, while the Tomcat and MySQL servers were hosted on AMD Athlon 64 3800+ machines with five operating frequencies. The machines were connected to each other using a 100Mbps Ethernet LAN. The plots show good linearity over the entire range of available

[1] UP: Univ. of Pennsylvania, UMD: Univ. of Maryland, DU: Duke Univ., PU: Princeton, MIT: Massachusetts Inst. of Tech., CU: Cornell Univ.

CPU frequencies despite the fact that only two frequencies were used to estimate the gradients.

C. VM Capacity Gradients

The final gradient we present is the *VM capacity gradient*. This gradient assumes a virtual machine environment in which each node is hosted in its own virtual machine (although several virtual machines may share a physical machine). The gradient is then defined as rate of change of the system's end-to-end transactional response time with respect to the fraction of CPU capacity allocated to each individual VM. The VM capacity gradient can be used to drive performance aware server consolidation and energy conservation. Specifically, it can help determine how much each virtual machine's CPU allocation can be reduced without violating response time SLAs so that VMs can be packed into the fewest number of physical hosts as possible.

We measure VM capacity gradients using a Xen 3.2 based environment, and use Xen's Dom0 configuration interface to dynamically adjust the CPU capacity allocated to each VM. In principle, VM capacity gradients are similar to frequency gradients and can thus use a similar basis function after appropriately scaling the CPU frequency f_n by the fractional CPU capacity c_n allocated to node n. However, the virtualization environment raises complications that must also be addressed. The most important difference is in the measurement of utilization U. For Xen based environments, the hypervisor acts as a conduit for all I/O requests from each VM to the physical hardware. In doing so, it acts on behalf of each VM, and thus the basis function requires the sum of the VM and hypervisor utilization when estimating the application constant α_n.

Using these modifications, the basis function achieves excellent linearity with respect to the measured end-to-end response time as can be seen from the x-y plot of Figure 2(d). The testbed used for this experiment consisted of Pentium 4 servers running a Xen 3.2 installation and communicating over a local 100Mbps Ethernet segment. To obtain the figure, the CPU allocation to the Tomcat server was varied from 25% to 60% and the values of the measured system response time were plotted against the values of the basis function for two different workloads. The figure shows that linearity is maintained despite the wide variation in CPU capacity and workload changes without the need for any model recalibration.

IV. Related Work and Conclusions

The general problem of performance prediction in multitier systems is a well-studied one. Queuing network formulations such as Layered queuing networks (LQN) [7] provide an especially appropriate formulation to model multitier systems and have been used effectively in many case studies. However, such models require detailed knowledge of the system transactions, their resource usage, and communication patterns. To alleviate these drawbacks, data intensive approaches including machine learning have also been proposed to construct models for black box systems as in [8] and [9]. However, such approaches can provide performance estimates only if very similar configurations have already been seen before, and thus do not provide true predictive capabilities. In contrast, by imposing restrictions on how a system's metrics evolve via basis functions formulated using high level knowledge about the behavior of distributed systems, gradients neither require detailed system knowledge nor extensive data collection. The closest related work is an approach proposed by Stewart et. al. in [10] and [11] in which passive data collection is combined with an M/M/1 queuing model based template in order to estimate the service and waiting times at each resource of a complex multitier system. However, their techniques become increasingly inaccurate as the system load increases. In contrast, by introducing active perturbations into a running system, our approach can predict metrics in configurations that are very different from those in which measurement was performed.

In conclusion, we believe that gradient based models are a practical, push-button alternative to traditional approaches for modeling complex multitier systems that rely heavily on domain knowledge. By applying gradient-based models to three different prediction problems, we have demonstrated the generality of the techniques and via experimental results, demonstrated their feasibility and accuracy. In future work, we intend to investigate gradients involving additional metrics and attributes, and to develop techniques for addressing unexpected and/or abrupt behaviors that can cause our basis function based techniques to fail. These include behaviors triggered due to timeouts and other application triggered adaptations.

References

[1] D. Galletta, R. Henry, S. McCoy, and P. Polak, "Web site delays: How tolerant are users?" *J. of the Assoc. for Information Sys.*, vol. 5, no. 1, pp. 1–28, 2004.

[2] I. Ceaparu, J. Lazar, K. Bessiere, J. Robinson, and B. Shneiderman, "Determining causes and severity of end-user frustration," *Int. Journal of Human-Computer Interaction*, vol. 17, no. 3, pp. 333–356, 2004.

[3] "The psychology of web performance," May 2008, accessed Apr. 2009. http://www.websiteoptimization.com/speed/tweak/ psychology-web-performance/.

[4] D. Farber, "Google's Marissa Mayer: Speed wins," ZDNet Between the Lines, Nov. 2006, accessed Apr. 2009. http://blogs.zdnet.com/BTL/?p=3925.

[5] S. Chen, K. Joshi, M. Hiltunen, W. Sanders, and R. Schlichting, "Link gradients: Predicting the impact of network latency on multitier applications," in *Proc. INFOCOM*, Apr. 2009.

[6] S. Chen, K. Joshi, M. Hiltunen, R. Schlichting, and W. Sanders, "Black-box prediction of the impact of DVFS on end-to-end performance of multi-tiered systems," in *GreenMetrics 2009 Workshop at SIGMETRICS 2009*, Jun. 2009.

[7] M. Woodside, J. Neilson, D. Petriu, and S. Majumdar, "The stochastic rendezvous network model for performance of synchronous client-server-like distributed software," *IEEE Trans. on Comp.*, vol. 44, no. 1, pp. 20–34, 1995.

[8] C. Stewart and K. Shen, "Performance modeling and system management for multi-component online services," in *Proc. Usenix NSDI*, vol. 2, 2005, pp. 71–84.

[9] P. Bodik, C. Sutton, A. Fox, D. Patterson, and M. Jordan, "Response-time modeling for resource allocation and energy-informed SLAs," in *Proc. Workshop on Stat. Learning Tech. for Solving Sys. Problems*, 2007.

[10] C. Stewart, T. Kelly, and A. Zhang, "Exploiting Nonstationarity for Performance Prediction," in *EuroSys*, March 2007.

[11] C. Stewart, T. Kelly, A. Zhang, and K. Shen, "A Dollar from 15 Cents: Cross-Platform Management for Internet Services," in *Proc. Usenix Annual Technical Conference*, June 2008.