Blackbox Prediction of the Impact of DVFS on End-to-End Performance 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

ABSTRACT

Dynamic voltage and frequency scaling (DVFS) is a well-known technique for gaining energy savings on desktop and laptop computers. However, its use in server settings requires careful consideration of any potential impacts on end-to-end service performance of hosted applications. In this paper, we develop a simple metric called the "frequency gradient" that allows prediction of the impact of changes in processor frequency on the end-to-end transaction response times of multitier applications. We show how frequency gradients can be measured on a running system in a push-button manner without any prior knowledge of application semantics, structure, or configuration settings. Using experimental results, we demonstrate that the frequency gradients provide accurate predictions, and enable end-to-end performance-aware DVFS for mulitier applications.

1. INTRODUCTION

DVFS is an energy-saving technique that has been extremely successful on laptop computers, and holds great promise for data center environments. The stakes are enormous. Data centers consumed 1.5% of all U.S. electrical power in 2006 [1], and their rate of energy use is rapidly growing. CPUs directly consume 30% of the total IT equipment power consumption within a data center [5], and indirectly affect much more by way of cooling requirements. Recognizing the potential benefits of DVFS, manufacturers such as Intel have recently started increasing the granularity and scope of DVFS control in server processors, such as in Xeon chips based on the Nehalem architecture.

However, DVFS has important ramifications for multitier applications that reside in these data centers and provide web-accessible services ranging from e-commerce to communication. Such services are often crucially dependent on quick response times to keep users satisfied. Recent studies [10, 6, 2] and the experiences of large operators [9] have shown the importance of end-to-end response time on user satisfaction, traffic growth, and, consequently, business viability. To ensure adoption for such applications, DVFS techniques will need to ensure that end-to-end responsiveness is maintained. However, current DVFS techniques, e.g., those employed in the Linux kernel (e.g., the *ondemand* power regulator), rely only on OS-level, single-machine measurements such as CPU utilization when making decisions, and ignore the impact of scaling on end-to-end application metrics.

In this paper, we tackle that problem by introducing a new metric, the frequency gradient, which quantifies the impact of frequency change on a machine on the end-to-end response time of a multitier application. For modern multitier applications, this impact may not be obvious. Changing the frequency of a server that is accessed asynchronously by user requests, e.g., a logging server, may not have any impact on user-perceived responsiveness. On the other hand, slowing down a bottleneck server may have a disproportionately large impact. These impacts can change as the bottlenecks and other dynamic characteristics of a system change, nevertheless, knowing the impacts enables smart DVFS by slowing down each host "just enough" to ensure that responsiveness constraints and/or service level agreements (SLAs) are met.

Frequency gradients provide a blackbox alternative to queuing models, which have been the traditional tool used to make response time predictions for multitier systems. While queuing models require detailed knowledge regarding the structure, resource usage characteristics, and communication patterns of the target system, frequency gradients provide a much simpler metric that is easy to measure online with very little knowledge of the target system. In return for their simplicity, gradients become increasingly inaccurate as the operating conditions move further from the measurement point. However, the push-button nature of measurement, coupled with techniques that we have developed to ensure low measurement overhead, ensure that the models can be recalibrated on-the-fly.

In previous work [8], we developed a similar metric, called the *link gradient*, to predict the impact of changes in network link latency on end-to-end application response times. The link gradient is a point derivative quantifying the rate of change of response time as a function of a change in link latency, and provides a linear predictor. We developed a spectral technique based on injection of small latency fluctuations followed by Fourier analysis to estimate link gradients at runtime with very low perturbation, while ensuring high accuracy even in noisy production environments.

In this paper, we extend the fundamental concepts developed with link gradients to frequency scaling, and make the following new contributions. a) We develop a metric and runtime measurement system to predict the impact of CPU frequency changes on the end-to-end response time of a multitier system. b) We enable the modeling of non-linear response time vs. CPU frequency relationships by formulating gradients as a linear combination of non-linear basis functions that are based on high-level, application-independent knowledge of system behavior. c) We enable the chaining of gradients at different levels of granularity. Specifically, we combine the impacts of changes in CPU frequency on individual nodes with the impacts of changes in individual nodes on the end-to-end behavior to produce end-to-end predictions of the impacts of CPU frequency changes. d) Finally, we demonstrate that the predictions made using frequency gradients are accurate regardless of application architecture, communication patterns, and configuration settings.

2. TECHNICAL APPROACH

2.1 Frequency Gradients

We consider a system consisting of a cluster of host machines $M = (m_1, m_2, \dots, m_m)$, each of which can be configured to operate at a frequency f_i from a range of available CPU frequencies. The target multitier application A is hosted on these machines, and consists of components $C = (c_1, c_2, \dots, c_n)$ of various types, such as web servers, application servers, or databases. Each machine may host one or more components. Users interact with the application 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. The overall workload w measured as the number of requests per second represents a mix of all the transactions.

The responsiveness of the system is measured in terms of the mean response time $\bar{rt}$, which is considered as an unknown function of the machine frequencies f_i and the application workload w. Let $\bar{rt}(x) = F(w, f_1, f_2, \dots, f_m)$ be this unknown function, where vector $x = (w, f_1, f_2, \dots, f_m)$ represents the operating configuration at which the measurement is made. Then, the question we wish to answer is, "Given the value of $\bar{rt}(x_0)$ at the system's current operating configuration x_0, what is the value of the response time at a different operating configuration x_1, i.e., $\bar{rt}(x_1)$?".

To answer this question, the general approach we take is as follows. Let the vector $\Delta x = x_1 - x_0 = (\Delta w, \Delta f_1, \dots, \Delta f_m)$ be the differential change in workload and host frequencies 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 response time as:

$$\bar{rt}(x_1) = \bar{rt}(x_0) + \sum_{i=1}^{m} \frac{\partial F}{\partial f_i}\bigg|_{x_0} \Delta f_i + \frac{\partial F}{\partial w}\bigg|_{x_0} \Delta w + O(\Delta x^2) \quad (1)$$

In this equation, the $O(\dots)$ term represents the higher order derivatives and powers of the frequencies and workload. If the derivatives $\frac{\partial F}{\partial x}\big|_{x_0} = (\frac{\partial F}{\partial w}\big|_{x_0}, \frac{\partial F}{\partial f_1}\big|_{x_0}, \dots, \frac{\partial F}{\partial f_m}\big|_{x_0})$ of the response time function (i.e., the gradients) are known at the current operating configuration x_0, one might imagine using this equation to predict the response time of the system in the new configuration by ignoring the higher-order derivatives and powers in $O(\Delta x^2)$. However, doing so is justifiable only if Δx is small enough to cause the higher powers to vanish, or if F is linear, thus ensuring that the higher-order derivatives are zero. Unfortunately, neither is true when predicting the effect of CPU frequency on response time. In practice, frequency and workload changes in machines can be quite large. Furthermore, because frequency scaling affects queuing behavior on each component, the CPU frequency-response time relationship can be nonlinear.

We solve the dilemma of nonlinearity using two techniques. First, we decompose the gradients into two sets of partial derivatives: the "system gradient" captures the rate at which the system's end-to-end response time changes with each per-component response time, while the "machine gradient" captures the rate at which each component's response time changes with its host machine's frequency and the workload. As motivated below, the system gradient is linear, while the machine gradient contains most of the nonlinearity in F. The second technique recasts the frequency gradients in terms of "basis functions" with respect to which the gradients are linear. Next, we describe both gradients in detail.

System Gradients. The *system gradient* is defined as a vector $(\partial\bar{rt}/\partial\bar{rt}_1, \dots, \partial\bar{rt}/\partial\bar{rt}_n)$ whose elements are the partial derivatives of the mean end-to-end response time of the system with respect to the mean response time of each component, i.e., $\bar{rt}_i$ in the system. Intuitively, this gradient is dependent on the call-graphs associated with user transactions. For example, a transaction that consists of a series of single nested calls to a web server, an application server, and a database would be expected to have a system gradient with all values equal to one. Communication patterns such as load balancing, caching, and state replication among servers impact the system gradient. However, the gradient is expected to remain constant under a range of operating configurations for a given application, thus indicating a linear relationship. In our previous work [8], we have experimentally demonstrated the linearity of a similar metric: link gradients that capture the derivative of end-to-end response time with respect to link latencies between the nodes.

Machine Gradients. Machine gradients capture the relationship between individual component response times and their host machines' CPU frequencies. Because this relationship can be nonlinear, we do not define machine gradients with respect to CPU frequency directly. Instead, we define them with respect to a basis function $\hat{f}_i(f_i, w)$ that encapsulates the nonlinearity by ensuring that the change in the component response time is linear with respect to the change in the basis function. Formally, the *machine gradient* for component c_i, or $\partial\bar{rt}_i/\partial\hat{f}_i$, is defined as the partial derivative of the component response time with respect to the basis function $\hat{f}_i$.

To select a suitable basis function $\hat{f}_i$, we rely on the observation that nonlinearity in the response time frequency relationship arises mainly from queuing and processor sharing effects. Therefore, we choose the response time function for a single M/G/1/PS (processor sharing) queue as the basis function. Specifically, the response time of such a queue is given by $rt = \frac{D/f}{1-U}$, where D is the mean processor demand generated by each request as measured in cycles, $U = a \cdot w/f$ is the processor utilization, and a is a transaction-specific constant. Using this relation, we define the basis function for node c_i as $\hat{f}_i(f, w) = \frac{1}{f_i - a_i w}$. Because the host frequency f_i, application workload w, and host utilization U_i can all be measured, our framework estimates the constant a_i by direct observation. Although the basis function was chosen based on high-level design information about the system, viz., that it uses a time-slice-based CPU scheduler, our approach is still a black-box one because application-specific parameters such as service time that require detailed instrumentation do not need to be estimated.

Frequency Gradients. In practice, measuring the system and machine gradients directly is difficult because we cannot measure or exert direct control over the per-component response times $\bar{rt}_i$. Therefore, we compose the system and machine gradients by using the chain rule of derivatives to form a composite *frequency gradient* that is not only easy to measure, but also capable of providing the complete relationship between end-to-end response time and per-host CPU frequencies. Based on the preceding discussion, we define the composite predictor as:

$$\bar{rt}(x_1) = \bar{rt}(x_0) + \sum_{i=1}^{n} \frac{\partial \bar{rt}}{\partial \bar{rt}_i} \frac{\partial \bar{rt}_i}{\partial \hat{f}_i} [\hat{f}_i(f_{i_1}) - \hat{f}_i(f_{i_0})] \tag{2}$$

Using this equation, the frequency gradient is defined as the vector $\left(\frac{\partial \bar{rt}}{\partial \bar{rt}_1} \frac{\partial \bar{rt}_1}{\partial \hat{f}_1}, \ldots, \frac{\partial \bar{rt}}{\partial \bar{rt}_n} \frac{\partial \bar{rt}_n}{\partial \hat{f}_n}\right)$, and can be measured directly by changing the frequency of each host machine, observing the changes in end-to-end response time and the basis functions, and computing the ratio.

Although the above techniques minimize the impact of nonlinearity to a large degree, large differences between the new and old configurations cause predictions made using frequency gradients to become increasingly inaccurate. Therefore, we develop gradient measurement techniques that are lightweight and minimally intrusive so that gradients can be recalculated at intermediate configurations cheaply and on-the-fly.

2.2 Signal Injection and Fourier Analysis

To measure the frequency gradient for a component, we used the signal injection and Fourier analysis technique proposed in [8]. Within a short time frame (usually several minutes), we inject a signal into the system by perturbing the CPU frequency of a server using a square wave pattern at a chosen frequency. The frequency is chosen such that the noise in that component of the frequency spectrum is small so as to improve the estimation accuracy. When the square wave is high, we lower the working CPU frequency, and when the square wave is low, we set the working CPU frequency to its default value. In this manner, we synthesize a square wave perturbation in $\hat{f}_i$. Finally, we use standard Fourier transforms to compute the frequency spectrum of the system's response time series during the period of signal injection, and use that frequency spectrum to estimate the frequency gradient. Use of this technique allows a reduction of noise and perturbation by an order of magnitude (see [8]) compared to a time-domain approach, and thus makes it possible to recalculate the gradients dynamically and cheaply while the system is running.

2.3 Online Measurement Framework

Using the basic approach described above, we have implemented a distributed active monitoring framework, shown in Figure 1, 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 machine. Each daemon is responsible for monitoring the CPU utilization of the server, reporting information about the server to the central coordinator, and changing the frequency of the server CPU when commanded by the central coordinator. To change the frequency of the server

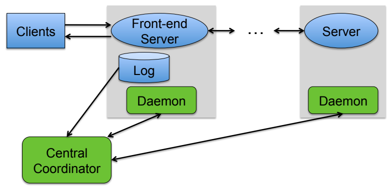


Figure 1: Monitoring architecture

Transaction	Apache	Tomcat	MySql
ViewUserInfo	0.743 (1.08)	5.79 (0.19)	1.87 (0.19)
ViewItem	0.36 (0.88)	4.82 (0.15)	2.03 (0.16)

Table 1: RUBiS Frequency Gradients (msec/GHz^{-1})

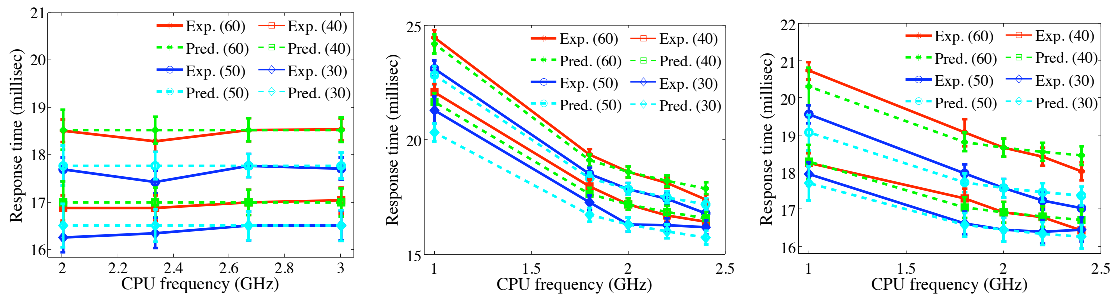
CPU at runtime, the local daemon uses the *CPUfreq* system interface enabled by the *userspace* CPU frequency scaling governor. It writes the desired CPU frequency to the *scaling_setspeed* interface, and reads the available scaling frequencies from the *scaling_available_frequencies* interface. To monitor the CPU utilization, we use *sar* to collect the CPU utilization periodically.

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 response time data. The framework supports applications that have web interfaces. The central coordinator parses the Apache access logs to extract a response time series—i.e., timestamp, end-to-end response time pairs—for each transaction's URL. No additional workload beyond the application's normal workload is required for measurement purposes, thus ensuring minimal interference with a running system.

The process of measuring the frequency gradients consists of two phases: a) the training phase, and b) a set of per-CPU measurement phases. In the training phase, the coordinator passively collects the system response times. It uses the response time series to determine the parameters for the delay square wave that is to be injected into each individual CPU. The per-CPU measurement phase is conducted once for every host machine, and is the active phase during which a perturbation is introduced into the target server and the system response is measured. The measurement technique is similar to the one used for measuring link gradients in [8].

3. EVALUATION

We provide a preliminary evaluation of our approach using RUBiS—a well-known eBay-like auction application [7]—widely used in multitier system benchmarks. We used the 3-tier Java-servlet version of RUBiS with a front-end Apache server (WS), two Tomcat application servers (TA and TB), and a back-end MySQL database server (DB). Our heterogeneous testbed consisted of 2 types of machines: an Intel E8400 Core 2 Duo machine that was used to run Apache and three AMD Athlon 64 3800+ machines for the Tomcat and MySQL instances. The Intel processor had 4 operating frequencies (2.0, 2.33, 2.67, and 3.0GHz) while the AMD chip had 5 frequencies (1.0, 1.8, 2.0, 2.2, 2.4 GHz). We emulated the client workload using multiple independent Poisson arrival processes in which the emulated users randomly submitted requests for information about an item or about another user with equal probability. The client emulator and the central coordinator were run on two AMD Athlon XP 2400 machines. All machines had 2GB of RAM, were connected using a 100Mbps Ethernet switch, and ran an unmodified Ubuntu Linux 8.04 installation.



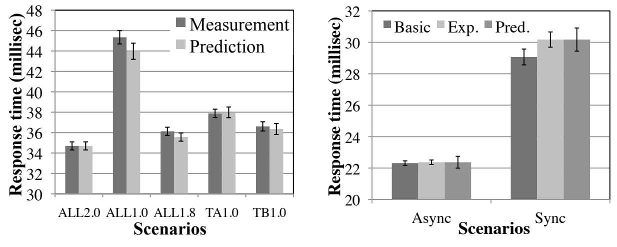
(a) Apache (b) Tomcat (c) MySQL

Figure 2: End-to-end response time predictions under workload and frequency changes

Using the measurement framework, we computed the frequency gradients in a baseline configuration with a single server in each tier, with all CPUs running at their third lowest frequency, i.e., 2.66GHz and 2.0GHz for the Intel and AMD CPUs respectively, and with a client workload of 40 req/sec. The values of the gradients for two different transaction types and all three servers are shown in Table 3 with standard deviations of the estimates shown in parentheses. Some observations are of note. First, the gradients depend on both the transaction and the server type, thus reflecting the different ways in which each transaction uses different types of servers. Based on the gradients, one can infer that both transactions are computationally intensive rather than database intensive (larger Tomcat gradient as compared to MySQL). Second, the standard deviation for the Apache gradient is very high. This is because load imposed on the Apache server by both transactions is very light, and any changes in its CPU frequency do not change the end-to-end transaction response time perceptibly, causing noise to dominate the measurement. Therefore, whenever the standard deviation is large, we infer that the gradient is small, and set it to zero.

Results of the predictions using the gradients are shown in Figure 2. In these experiments, we use Equation 2 to predict the end-to-end mean response time of the system, averaged across all transactions, under different workloads, and with a frequency change in a single CPU. Then, we deployed the changed configuration and compared the prediction against the measured end-to-end response time. Each graph reflects results across a range of workloads (30-60 req/sec) for a single CPU frequency change in the server indicated by the graph's title. The graphs show that the difference between the prediction and measurements increases as the frequency moves further away from the measurement configuration (2.66GHz for the Apache server, and 2.0GHz for the Tomcat and MySQL servers). Nevertheless, in all cases, the 95% confidence intervals for the predicted response time values overlap with the 95% confidence intervals for the measured results, thus indicating that the frequency gradient model provides good agreement with experiment. The confidence intervals associated with the predictions are due to the standard deviation in the frequency gradient measurements shown in Table 3. In order to increase prediction accuracy, the confidence intervals could be reduced by increasing the length of the gradient measurement phase. Our setting of the Apache frequency gradient to zero because of excessive variance is also validated by the results, which show no appreciable change in the transaction end-to-end response time due to changes in the Apache server's CPU frequency.

Next, to show the blackbox applicability of the frequency



(a) Load balancing (b) State replication

Figure 3: Different communication patterns

gradient metric when application configuration and characteristics change, we show experimental results under two common deployment configurations. In the first configuration, we deployed two Tomcat servers in a non-replicated load-balanced configuration using Apache's *mod_jk* module. We measured the frequency gradients in a base setup with the same frequencies as the previous experiment, but with a larger client workload of 80 req/sec to account for the increased capacity of the system. Then, we changed the frequencies of multiple CPUs at a time and used the frequency gradients to predict the end-to-end transaction response times in the new settings. Finally, we deployed the system to run at the new frequencies and measured the actual mean end-to-end response time across all transactions to compare against the predictions. Figure 3(a) shows the predicted and experimental mean end-to-end response time results for several frequency settings. In the configurations ALL1.0, ALL1.8, and ALL2.0, we set the frequencies for all AMD CPUs to their lowest (1.0 GHZ), second lowest (1.8 GHz), and third lowest values (2.0 GHz), respectively. In the configurations TA1.0 and TB1.0, we set the frequencies of the Tomcat A and B servers to their lowest value (1.0 GHz), respectively. Figure 3(a) shows that the predicted vs. measured response times show good agreement.

For the final set of experiments, we configured the two Tomcat servers in a primary-backup replicated session-state setup using two different replication settings. In the synchronous setting, the primary updates the backup server before replying to the client, while in the asynchronous setting, the update is performed asynchronously after a reply to the client is sent by the primary. Since RUBiS does not normally use session state, we modified the transactions to exercise this facility. Then, we computed frequency gradients for the backup Tomcat server under both settings with a frequency of 2.0GHz and a workload of 100req/sec, and compared the gradient predictions of end-to-end response time vs. experimental measurements when the frequency of the backup Tomcat server was changed to 1.0GHz. The end-to-end response time in the original measurement configuration, the frequency gradient predictions, and the corresponding experimental measurements are shown in Figure 3(b). As expected, there is no impact of the backup server frequency change on the end-to-end response time in the asynchronous replication setting because the primary replies to the client before it updates the backup. In the synchronous setting however, a change in response time can be seen. Nevertheless, in both settings, the frequency gradients can predict the impact of the frequency change accurately.

Collectively, these experiments demonstrate the ability of the frequency gradient metric to accurately predict the impact of DVFS on end-to-end application response time in a blackbox manner under varying frequencies, workloads, and application settings.

4. RELATED WORK

To the best of our knowledge, [11] is the only work that considers end-to-end performance impact when performing DVFS for multitier applications. This work assumes a pipelined, and uses a traditional M/M/1 queuing network model for performance prediction. To obtain the parameters for the model, server instrumentation along with offline profiling is used. In contrast, our work is not limited to pipelined systems as demonstrated by the Tomcat replication results, and is a blackbox approach that does not require knowledge of application topology, configuration, or instrumentation of server components. Independently of DVFS, the general problem of performance prediction in multitier systems is a well-studied one. In [4, 12], traditional modeling methods are used to associate performance metrics (i.e., response times) with variables representing the workload being processed, the machines' architecture, and the communication patterns. The profile is then used to predict the system performance during deployment. [3] and [13] use queuing networks and models to estimate the end-to-end response time of multitier Internet applications. Layered queuing networks (LQN) [14] provide an especially appropriate formulation to model multitier systems. However, all of these methods either require offline profiling of the system to compute the necessary parameters and/or a detailed knowledge of the system architecture and configuration settings.

5. CONCLUSIONS

DVFS is a well-known technique for reducing the power usage of a computer by slowing down the CPU. However, determining what frequency to use so that a multitier application's response time guarantees are still met is a challenging problem and cannot be solved simply by adjusting frequency based on local utilization at each machine. In this paper we have presented a simple measurement-based technique that can be used to predict the response time of a multitier application for any CPU frequency assignments in the system. It is based on lightweight runtime measurements that capture the impact of CPU frequency changes on the end-to-end response time of the system. We demonstrated the accuracy of these predictions using a number of application deployment scenarios. Our future work will include development of a runtime controller that optimizes the energy usage of a data center based on the frequency gradient.

6. REFERENCES

[1] EPA Report on Server and Data Center Energy Efficiency, Aug 2007. http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf.

[2] The psychology of web performance, May 2008. http://www.websiteoptimization.com/speed/tweak/psychology-web-performance/.

[3] Bhulai, S., Sivasubramanian, S., van der Mei, R., and van Steen, M. Modeling end-to-end response times in multitier Internet applications. *Man. Traffic Perf. in Converged Networks 4516* (2007), 519–532.

[4] Bodik, P., Sutton, C., Fox, A., Patterson, D., and Jordan, M. Response-time modeling for resource allocation and energy-informed SLAs. In *Proc. Workshop on Statistical Learning Techniques for Solving Systems Problems (MLSys)* (2007).

[5] Castro-Leon, E. Musings about data center energy usage, Sept 2008. http://communities.intel.com/openport/community/openportit/ipip/blog/2008/09/07/musings-about-data-center-energy-usage.

[6] Ceaparu, I., Lazar, J., Bessiere, K., Robinson, J., and Shneiderman, B. Determining causes and severity of end-user frustration. *Int. Journal of Human-Computer Interaction 17*, 3 (2004), 333–356.

[7] Cecchet, E., Marguerite, J., and Zwaenepoel, W. Performance and scalability of EJB applications. In *Proc. OOPSLA'02* (2002), pp. 246–261.

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

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

[10] Galletta, D., Henry, R., McCoy, S., and Polak, P. Web site delays: How tolerant are users? *J. of the Assoc. for Information Sys. 5*, 1 (2004), 1–28.

[11] Horvath, T., Abdelzaher, T., Skadron, K., and Liu, X. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Trans. on Comp. 56*, 4 (July 2006), 444–458.

[12] Stewart, C., and Shen, K. Performance modeling and system management for multi-component online services. In *Proc. NSDI* (2005), vol. 2, pp. 71–84.

[13] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., and Tantawi, A. An analytical model for multi-tier internet services and its applications. In *Proc. ACM SIGMETRICS* (2005), pp. 291–302.

[14] Woodside, M., Neilson, J., Petriu, D., and Majumdar, S. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Trans. on Comp. 44*, 1 (1995), 20–34.