Empirical Examination of A Collaborative Web Application

Christopher Stewart Matthew Leventi Kai Shen
Department of Computer Science, University of Rochester
{stewart, mleventi, kshen}@cs.rochester.edu

Abstract

Online instructional applications, social networking sites, Wiki-based web sites, and other emerging web applications that rely on end users for the generation of web content are increasingly popular. However, these collaborative web applications are still absent from the benchmark suites commonly used in the evaluation of online systems. This paper argues that collaborative web applications are unlike traditional online benchmarks, and therefore warrant a new class of benchmarks. Specifically, request behaviors in collaborative web applications are determined by contributions from end users, which leads to qualitatively more diverse server-side resource requirements and execution patterns compared to traditional online benchmarks. Our arguments stem from an empirical examination of WeBWorK — a widely-used collaborative web application that allows teachers to post math or physics problems for their students to solve online. Compared to traditional online benchmarks (like TPC-C, SPECweb, and RUBiS), WeBWorK requests are harder to cluster according to their resource consumption, and they follow less regular patterns. Further, we demonstrate that the use of a WeBWorK-style benchmark would probably have led to different results in some recent research studies concerning request classification from event chains and type-based resource usage prediction.

1 Introduction

Traditional web applications service requests from content supplied by some central content generation sources (in the form of static web pages or dynamic content produced by hosted content generation code). Although some user requests may result in persistent state changes on the server side, such changes often exert very small effects on system-level application execution behaviors. Due to the large dependence on central content generation sources in these applications, the resource requirements and other characteristics for processing user requests are typically well clustered into small numbers of groups [3, 6, 15]. Widely used online benchmarks such as TPC benchmarks (*e.g.*, TPC-C, TPC-H, TPC-W) [22], SPEC online benchmarks (*e.g.*, SPECweb, SPECjbb, SPECjAppServer) [16], and J2EE-based multi-component benchmarks (*e.g.*, RUBiS [13], Stock-Online [21]) all reflect such request behavior patterns.

Recent years have witnessed an emerging class of web applications that rely on end users for their content. In some cases, these sites even rely on end users for code that generates content. Examples of such applications include social networking platforms [8], web-based productivity software [9], online instructional applications [24], and Wiki-based web sites [7]. The popularity and vitality of these *collaborative web applications* directly benefit from an ever-increasing pool of creative web users. At the same time, due to the many contributions from independent end users, the behavior pattern of request processing in these applications tends to be much less clustered or regular. Despite the surging importance of these applications and their behavior uniqueness, they have been ignored in the benchmarking of online web applications.

This paper makes the case for a new class of benchmarks that reflect the unique properties of collaborative web applications. Our arguments stem from an empirical examination of WeBWorK [24], a real-world collaborative web application. WeBWorK is a web-based homework checker that allows teachers to post math or physics problems for their students to solve online. In particular, teacher-supplied WeBWorK problems are interpreted by the application server as content-generating scripts. The WeBWorK deployment at the University of Rochester is used by around 50,000 students from 80 or so institutions worldwide. Our empirical examination is driven by realistic problem sets (ranging from pre-calculus to differential equations) and user requests extracted from three-year system logs at the real site. We emphasize that the use of *realistic* workload traces is particularly important in this study because the user-supplied content may substantially affect the request processing behavior pattern in WeBWorK.

As a representative collaborative web application, WeBWorK provides a realistic basis to reevaluate past research findings derived on traditional online benchmarks. In particular, this paper reevaluates two recent contributions on the system-level management of online services: 1) Magpie-style online request classification which is based on canonical request event chains [3]; 2) system resource usage prediction based on a linear request-type model [19].

This paper makes two contributions.

1. We present an empirical evaluation that exposes fundamental differences between a real-world collaborative web application and traditional benchmarks commonly used to evaluate web applications.

2. We demonstrate that the use of a collaborative web application as an evaluation basis would probably have led to

different conclusions in some recent research studies.

This remainder of this paper is organized as follows: Section 2 provides more background on WeBWorK and collaborative web applications. Section 3 contrasts the request behavior characteristics of WeBWorK against traditional online benchmarks. Section 4 uses WeBWorK as an evaluation basis for two recent research proposals, and finds that it may lead to different conclusions compared to using traditional online benchmarks. Section 5 describes related work and Section 6 concludes. We also announce the release of our WeBWorK trace, dataset, and deployment instructions at the end of this paper.

2 WeBWorK and Collaborative Web Applications

WeBWorK [24] supports two types of users. Teachers create new problem sets to the online database. Students view problems and submit solutions. Solution submission also triggers online solution checking. In WeBWorK, the main content (problem sets and solution checkers) is generated collaboratively by a large number of teachers. In a three-year usage log, we have observed more than 3,000 teacher-created problem sets (which differ significantly from each other). The delivery of different problem sets and corresponding solution checkers may exhibit very different request processing behaviors at the server system.

In addition to the differing teacher-created problem sets, a single problem may also be delivered to multiple students in different ways. More specifically, WeBWorK allows teachers to create problems in the form of dynamic code (written in a Perl variant called the *Problem Generation (PG) Language*). An example of PG code is given in Figure 1. Such code is executed when a student accesses or submits a solution for a homework problem. The power of dynamically producing questions and correcting answers offers a practical benefit to teachers: the same core problem (*e.g.*, integration or linear equation) can be presented with some randomized variations to minimize the chance of direct solution copying among students.

WeBWorK distinguishes between users that can create content (teachers) and those that can view content only (students). Such dichotomies allow service providers to selectively revoke content generation privileges, and are common among commercial collaborative web applications. The Facebook Platform [8] and Google Application Engine [2], for instance, require end users to register as developers before they are allowed to generate content. Even Wiki-based applications, one of the original domains that allow any user to create or modify content, are increasingly password protected.

WeBWorK users are allowed to provide dynamically executing code for content generation. Other popular collaborative web applications also allow end users to create scripts that produce web content. For example, the Facebook Platform [8] invokes user-created scripts on remote machines and provides a well-defined API for such applications to access proprietary data. As another example, spreadsheet data processed

```
TEXT(beginproblem());
$showPartialCorrectAnswers = 0;

$x=random(-20,20,1);
$y=random(-20,20,1);
$z=random(-20,20,1);

$cox1 = non_zero_random(-5,5,1);
$coy1 = non_zero_random(-5,5,1);
$coz1 = non_zero_random(-5,5,1);
$cox2 = non_zero_random(-5,5,1);
$coy2 = non_zero_random(-5,5,1);
$coz2 = non_zero_random(-5,5,1);
$cox3 = non_zero_random(-5,5,1);
$coy3 = non_zero_random(-5,5,1);
$coz3 = non_zero_random(-5,5,1);

$b1 = $cox1*$x + $coy1*$y + $coz1*$z;
$b2 = $cox2*$x + $coy2*$y + $coz2*$z;
$b3 = $cox3*$x + $coy3*$y + $coz3*$z;

BEGIN_TEXT
Use Cramer''s rule to find the value of \(z\) in the
solution of the following system:
\[
\begin{array}{r}
$cox1 x ? {$coy1} y ? {$coz1} z = $b1 \\\\
$cox2 x ? {$coy2} y ? {$coz2} z = $b2 \\\\
$cox3 x ? {$coy3} y ? {$coz3} z = $b3 \\\\
\end{array}
\]
$BR
\(z=\) \{ans_rule(25) \}
$BR
END_TEXT

$ans3 = $z;

ANS(num_cmp($ans3));
```

Figure 1: An example PG code segment — solving a system of linear equations.

by Google Docs [9] can spur server-side execution of user-created calculations and graphs. Finally, the Second Life [14] virtual world allows users to create virtual items with dynamic, compute-intensive properties.

WeBWorK is structurally similar to many collaborative web applications, but it is not representative of all. We use WeBWorK as a case study to advocate a new research agenda on benchmarking collaborative web applications. WeBWorK is ideal for this purpose, because it is real and widely-used, and we have made its workload traces publicly available [23]. Therefore, our analysis is based on real usage patterns for an important application, and our results can be reproduced for verification.

3 Request Behavior Analysis

We examine WeBWorK's request behavior characteristics and contrast it with traditional online benchmarks. Here we employ several widely used online benchmarks for the purpose of comparison:

- *TPC-C* [22] simulates a population of terminal operators executing Order-Entry transactions against a database. It contains five types of transactions: "new order", "payment", "order status", "delivery", and "stock level", con-

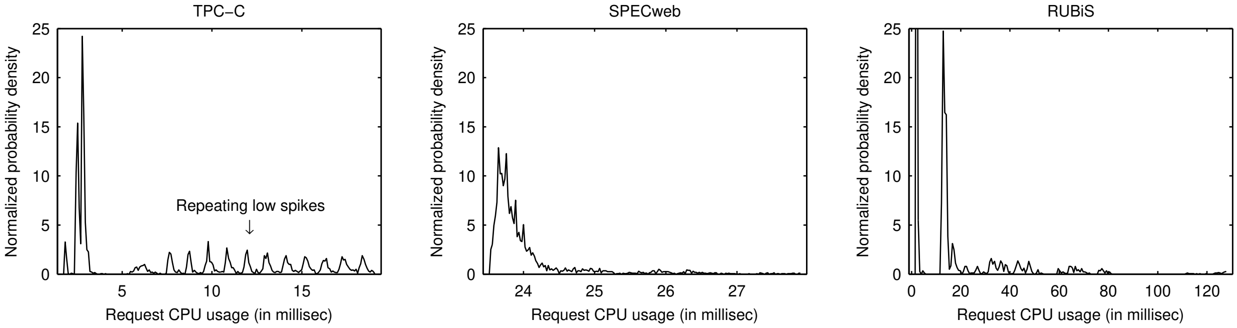

Figure 2: Probability density distribution of per-request CPU usage for three traditional online benchmarks. The probability density is normalized to that under the even distribution. To avoid distortion due to rare outliers requests, we consider a *viable* range of request CPU usage from the 1-percentile request CPU usage to the 99-percentile request CPU usage for each benchmark (X-axis value range in each plot).

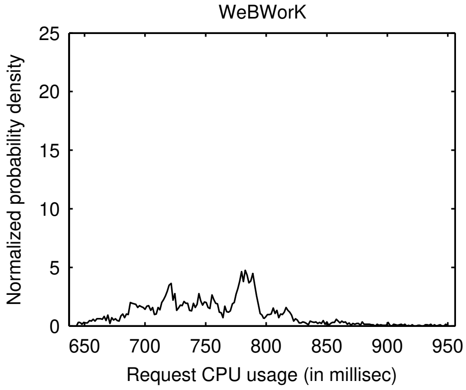

Figure 3: Probability density distribution of per-request behavior for WeBWorK. Refer to the caption of Figure 2 for details on setup.

stituting 45%, 42%, 4%, 4%, and 4% of all requests, respectively. We utilize a local implementation of TPC-C that may not follow all benchmark performance reporting rules. This is sufficient for our purpose of application behavior characterization.

- We use an assortment of dynamic content requests in the *SPECweb*99 benchmark [16]. Following the benchmark specification, we use a mix of 42% GET requests without cookie, 42% GET requests with cookie, and 16% POST requests.

- *RUBiS* [13] is a J2EE-based multi-component online service that implements the core functions of an auction site including selling, browsing, and bidding. It uses a three-tier service model, containing a front-end web server, a back-end database, and nine business logic components implemented as Enterprise Java Beans.

For our empirical evaluation, we deploy WeBWorK and the other three benchmarks in an experimental platform with system-level event tracing. In our deployments, WeBWorK, TPC-C, and RUBiS run on the MySQL 5.0.18 database. SPECweb employs the Apache 2.0.44 web server. Additionally, RUBiS runs on the JBoss 3.2.3 application server with an embedded Tomcat 5.0 servlet container. The server machine in our experimental platform contains dual 2 GHz Intel Xeon processors and 2 GB memory. All deployed application/benchmarks are CPU-bound. The server operating system is Linux 2.6.10 with an augmented request context maintenance and event tracing framework [15]. This framework allows us to collect per-request event traces for multi-component server applications. In particular, we have collected per-request traces on CPU context switch events, system call events, network and storage I/O events. The CPU context switch event trace allows us to derive per-request CPU usage for each application/benchmark.

Figures 2 and 3 plot the probability density distribution of per-request CPU usage for the three traditional online benchmarks and for WeBWorK respectively. The probability density is normalized to that under the even distribution. A visual examination uncovers two important behavior differences between WeBWorK and traditional online benchmarks:

- **Weaker clustering**. Per-request CPU usage probability density plots for traditional online benchmarks are strongly dominated by sharp spikes (or request behavior tends to form clusters). In contrast, WeBWorK exhibits much less clustered per-request CPU usage.

- **Less regularity**. Request behaviors for traditional online benchmarks sometimes exhibit regular patterns due to certain artificial effects of central content generation sources. For instance, the repeating low spikes in the TPC-C plot is due to an item count variable that is randomly selected within $\{5, 6, \cdots, 15\}$ for each "new order" transaction (according to Clause 2.4.1.3 of the TPC-C specification [22]). In contrast, WeBWorK requests do not exhibit any obvious regular patterns.

After examining request behavior patterns on a single request property (CPU usage), we next examine the correlation between multiple request properties. Figures 4 and 5 illustrate such correlation between request CPU usage and request system call count for the three traditional online benchmarks and for WeBWorK respectively. A visual examination uncovers another difference between them:

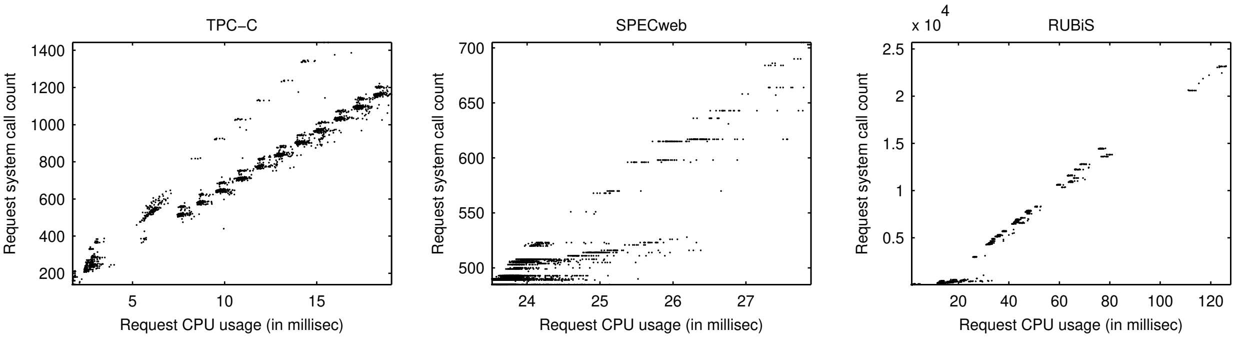

Figure 4: Two-dimensional plots on request CPU usage and request system call count for the three traditional online benchmarks. Each dot represents a request. Around 4,000 requests are illustrated for each benchmark.

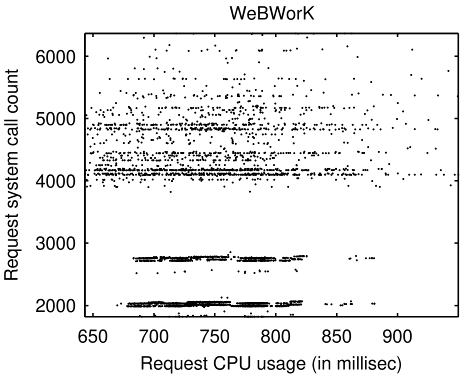

Figure 5: Two-dimensional plot on request CPU usage and request system call count for WeBWorK. Each dot represents a request. 4,000 requests are illustrated in the plot.

- **Weaker inter-property correlation**. For TPC-C and RUBiS, the correlation is visually very strong — the request CPU usage typically falls into a small value range when the request system call count is known. Although such correlation is weaker for SPECweb, it is still much stronger than that for WeBWorK.

Fundamentally, these unique behavior characteristics of WeBWorK can all be rationalized given that the behavior pattern of request processing in collaborative applications is heavily affected by user creation. Independent contribution from large numbers of end users injects a sense of randomness in request behaviors that lead to weaker request clustering, less behavior regularity, and weaker inter-property correlation.

4 Reevaluation of Past Findings

Differences on inherent benchmark characteristics do not necessarily lead to varying high-level conclusions from benchmark-driven evaluations [17]. In this section, however, we examine how the unique behavior of collaborative web applications may affect the evaluation results. By revisiting some recent research studies evaluated using traditional online benchmarks, we demonstrate that the use of a WeBWorK-style benchmark would probably have led to negative results.

4.1 Event Chain-based Request Classification

The Magpie work [3] proposed to transparently extract per-request event traces online and then use request event chains as signatures to classify similar requests into clusters. Finally, a concise workload model consists of each cluster's representative request and its relative size.

We implemented Magpie's request classification algorithm and applied it on the system call event trace for all four application/benchmarks. In Magpie, the distance of two requests is measured using the string-edit distance [10] of two requests' event chains. The calculation of string-edit distance is expensive for some of our application/benchmarks (both RUBiS and WeBWorK contain requests with tens of thousands of system call events). We use an alternative distance measure that can be quickly calculated — ignoring the event order and considering each request event chain as a counting set of events[1]; then calculating the minimum number of additions/deletions to equalize the two counting sets.

We evaluate the request classification accuracy by measuring how well each cluster representative's CPU usage predicts the CPU usage of all cluster members. We use the coefficient of determination (or R^2) as the model prediction accuracy metric. Specifically, given a set of samples $(x_1, x_2, \cdots, x_n)$ with sample mean $\bar{x}$ and corresponding predictions $(\hat{x_1}, \hat{x_2}, \cdots, \hat{x_n})$, the coefficient of determination for the prediction is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(x_i - \hat{x_i})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Intuitively $1 - R^2$ represents the relative aggregate square error of the prediction compared to that under the mean value prediction. A larger coefficient of determination (which cannot exceed 1.0) indicates more accurate prediction. A negative coefficient of determination indicates the prediction is less accurate than simply using the mean value prediction.

Table 1 compares the request classification accuracy evaluated using all four application/benchmarks when around 20

[1] In a counting set, we not only account for the system call types appearing in the set, but also the number of events for each type.

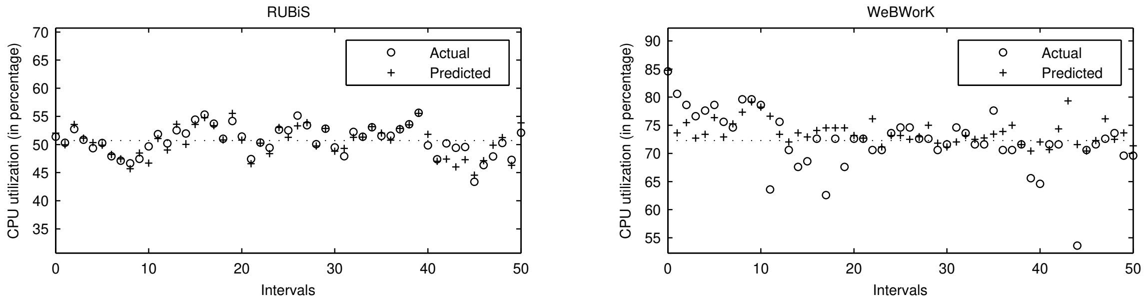

Figure 6: Type-based resource usage prediction over 50 five-minute intervals for RUBiS and WeBWorK. The dotted line in each plot indicates the actual mean.

Benchmark	R^2 accuracy
TPC-C	0.98
SPECweb	0.94
RUBiS	0.89
WeBWorK	−0.24

Table 1: Event chain-based request classification accuracy when the evaluation is driven by different benchmarks.

Benchmark	R^2 accuracy
RUBiS	0.90
WeBWorK	0.25

Table 2: Type-based resource usage prediction accuracy when the evaluation is driven by different benchmarks.

clusters are formed. Results suggest that the request classification is very accurate for the three traditional online benchmarks (within which TPC-C was actually used in the Magpie study [3]). In contrast, the WeBWorK-driven evaluation shows very poor request classification accuracy. This result is not surprising given our request behavior analysis in Section 3.

4.2 Type-based Resource Usage Prediction

Another recent work [19] analyzed real traces from production web applications and uncovered an interesting phenomenon: the relative frequencies of request types fluctuate over short and long intervals. Such nonstationarity can be used to calibrate models of application-level performance using only logs of request arrivals and resource utilizations that are routinely collected in production environments. One key submodel in this approach is a weighted linear characterization of aggregate CPU utilization shown below:

$$U = \beta_0 + \sum_j \beta_j N_j \tag{1}$$

where β_j represents the typical CPU demand of request type j (β_0 indicates the background CPU utilization not tied to specific request processing), and N_j is the number of requests of type j occurring in a particular time interval (*e.g.*, five minutes).

Using the trace-driven WeBWorK, we reevaluated the linear request-type model to predict system resource utilization. We consider three natural request types for WeBWorK: submitting problems, viewing problems, and submitting solutions. Table 2 shows that the request-type model yields high prediction accuracy for RUBiS, as reported in the previous work [19]. However, the prediction accuracy for WeBWorK is much worse. Figure 6 graphically depicts the prediction accuracy over 50 five-minute intervals for the two benchmarks, which clearly shows the poor accuracy for WeBWorK. Again, this result is not surprising given our request behavior analysis in Section 3.

5 Related Work

Previous benchmarks reflect the behavior patterns of traditional web workloads. RUBiS and RUBBoS [1] characterize the workload of a dynamic-content auction site and bulletin board, respectively. TPC benchmarks [22] reflect e-commerce and database workloads, and the SPEC suite [16] characterizes workloads for static content and multi-tier services. In comparison, WeBWorK captures the unique and emerging workload of a collaborative web application. Specifically, request execution patterns in WeBWorK are qualitatively more diverse compared to previous benchmarks, because they depend on the contributions of end users.

Recent studies have characterized other emerging web workloads. Nagpurkar *et al.* [12] investigate the instruction and cache miss behavior of a Web 2.0 blog, a social bookmarking site, and model-view-controller (MVC) PetStore. Lim *et al.* [11] explore new architectural designs for datacenter servers using interactive web mail and Mapreduce benchmarks. Cha *et al.* [4] find that user-supplied content affects the popularity distributions of web objects in YouTube, another real-world collaborative web application. Our contribution is the characterization of diversity in the server-side resource requirements and request executions patterns of WeBWorK.

Finally, it is well known that Internet services—of all varieties—are hard to manage. Stewart *et al.* demonstrate server consolidation [19], power-aware platform selection [18], and capacity planning [20] for dynamic-content Internet services. Chen *et al.* [5] demonstrate energy-aware server provisioning for a connection-intensive services, like video streaming and instant messaging. Barham *et al.* [3] demonstrate per-request cost accounting and anomaly detection for services distributed across a cluster. This paper shows that the workload of a collaborative web application is qualitatively unlike the workloads used by previous studies, and may require different solutions.

6 Conclusion

In this paper, we empirically examined a real-world collaborative web application — WeBWorK [23]. Our study finds that WeBWorK requests exhibit much weaker request clustering, less behavior regularity, and weaker inter-property correlation compared to traditional online benchmarks. All these behavior characteristics can be attributed to the independent content creation from large numbers of end users in emerging collaborative web applications [7–9, 14]. Using WeBWorK, we reevaluated some recent research findings (concerning event chain-based request classification [3] and type-based resource usage prediction [19]) and discovered that the use of a WeBWorK-style benchmark would probably have led to different results. We expect that the importance of collaborative web applications will rapidly rise given their surging popularity and vitality through direct user creation. The goal of this work is to raise awareness on the need for collaborative web application benchmarks, particularly for the purpose of evaluating computer system support for online services [3, 6, 15, 18–20].

Availability Our WeBWorK setup, including the request trace and dataset from the University of Rochester, is publicly available [23]. We hope other researchers will use WeBWorK to evaluate their research proposals on a collaborative web application workload.

Acknowledgments Professor Michael Gage provided the trace and dataset from the WeBWorK deployment at the University of Rochester. Students in the *Advanced Operating Systems* course at the University of Rochester gave early feedbacks on our characterization of the WeBWorK workload. Recommendations from anonymous reviewers helped us pinpoint the contributions of this paper. This work was supported in part by the U.S. National Science Foundation (NSF) grants CNS-0615045, CCF-0621472, NSF CAREER Award CCF-0448413, and by an IBM Faculty Award.

References

[1] C. Amza, A. Chanda, E. Cecchet, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *IEEE Workshop on Workload Characterization*, November 2002.

[2] Google app engine. http://code.google.com/appengine.com.

[3] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modeling. In *OSDI*, San Francisco, CA, Dec. 2004.

[4] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system. In *IMC*, San Diego, CA, Oct. 2007.

[5] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhang. Energy-aware server provisioning and load dispatching for connection-intensive Internet services. In *NSDI*, San Francisco, CA, Apr. 2008.

[6] M. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based failure and evolution management. In *NSDI*, San Francisco, CA, Mar. 2004.

[7] W. Cunningham and B. Leuf. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley, 2001.

[8] Facebook social network site. http://www.facebook.com.

[9] Google Docs. http://documents.google.com.

[10] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10, 1966.

[11] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *ISCA*, Bejing, China, June 2008.

[12] P. Nagpurkar, W. Horn, U. Gopalakrishnan, N. Dubey, J. Jann, and P. Pattnaik. Workload characterization of selected JEE-based web 2.0 applications. In *IISWC*, Seattle, WA, Sept. 2008.

[13] RUBiS: Rice University Bidding System. http://rubis.objectweb.org.

[14] Second Life. http://www.secondlife.com.

[15] K. Shen, M. Zhong, S. Dwarkadas, C. Li, C. Stewart, and X. Zhang. Hardware counter driven on-the-fly request signatures. In *ASPLOS*, Seattle, WA, Mar. 2008.

[16] Standard Performance Evaluation Corporation. SPEC benchmarks. http://www.spec.org.

[17] R. Stets, K. Gharachorloo, and L. Barroso. A detailed comparison of two transaction processing workloads. In *IEEE Workshop on Workload Characterization*, Austin, TX, Nov. 2002.

[18] C. Stewart, T. Kelly, K. Shen, and A. Zhang. A dollar from 15 cents: Cross-platform management for internet services. In *USENIX*, Boston, MA, June 2008.

[19] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *Eurosys*, Lisbon, Portugal, Mar. 2007.

[20] C. Stewart and K. Shen. Performance modeling and system management for multi-component online services. In *NSDI*, Boston, MA, May 2005.

[21] The StockOnline benchmark. http://forge.objectweb.org/projects/stock-online.

[22] Transaction Processing Performance Council. TPC bencmarks. http://www.tpc.org.

[23] WeBWorK benchmark: Datasets, traces, and installation instructions. http://www.cs.rochester.edu/u/stewart/collaborative.html.

[24] WeBWorK: Online homework for math and science. http://webwork.maa.org.