R-Capriccio: A Capacity Planning and Anomaly Detection Tool for Enterprise Services with Live Workloads

Qi Zhang[1], Ludmila Cherkasova[2], Guy Mathews[2], Wayne Greene[2], and Evgenia Smirni[1]

[1] College of William and Mary, Williamsburg, VA 23187, USA*
{qizhang,esmirni}@cs.wm.edu
[2] Hewlett-Packard Laboratories, Palo Alto, CA 94304, USA
{lucy.cherkasova,guy.mathews,wayne.greene}@hp.com

Abstract. As the complexity of IT systems increases, performance management and capacity planning become the largest and most difficult expenses to control. New methodologies and modeling techniques that explain large-system behavior and help predict their future performance are now needed to effectively tackle the emerging performance issues. With the multi-tier architecture paradigm becoming an industry standard for developing scalable client-server applications, it is important to design effective and accurate performance prediction models of multi-tier applications under an enterprise production environment and a real workload mix. To accurately answer performance questions for an existing production system with a real workload mix, we design and implement a new capacity planning and anomaly detection tool, called *R-Capriccio*, that is based on the following three components: *i)* a *Workload Profiler* that exploits locality in existing enterprise web workloads and extracts a small set of most popular, core client transactions responsible for the majority of client requests in the system; *ii)* a *Regression-based Solver* that is used for deriving the CPU demand of each core transaction on a given hardware; and *iii)* an *Analytical Model* that is based on a network of queues that models a multi-tier system. To validate *R-Capriccio*, we conduct a detailed case study using the access logs from two heterogeneous production servers that represent customized client accesses to a popular and actively used HP Open View Service Desk application.

1 Introduction

As IT and application infrastructures become more complex, predicting and controlling the issues surrounding system performance and capacity planning become a difficult and overwhelming task. For larger IT projects, it is not uncommon for the cost factors

* This work was largely completed in the summer of 2006 when Qi Zhang did an internship at HPLabs. E. Smirni are supported in part by the National Science Foundation (ITR-0428330). Currently, Qi Zhang is employed by MicroSoft and can be reached at the following address: qizha@microsoft.com. E. Smirni has been partially supported in part by the National Science Foundation under grants ITR-0428330 and CNS-0720699, and by a gift from Hewlett-Packard.

related to performance tuning, performance management, and capacity planning to result in the largest and least controlled expense. Application performance issues have an immediate impact on customer satisfaction. A sudden slowdown can affect a large population of customers, can lead to delayed projects, and ultimately can result in company financial loss. It is not unusual for a piece of new hardware to be added into the infrastructure to alleviate performance issues without fully understanding where the problem really is.

With complexity of systems increasing and customer requirements for QoS growing, the research challenge is to design an integrated framework of measurement and system modeling techniques to support performance analysis of complex enterprise systems in order to explain large-system behavior. Predicting and planing future performance is of paramount importance for the commercial success of enterprise systems.

Large-scale enterprise development projects are relying more and more on the *Service-Oriented Architecture* (SOA) design. This approach provides a collection of mechanisms and interfaces for a dynamic enterprise IT environment to connect applications where classic, data-processing legacy systems can be integrated with agile, web-based front-end applications. Application servers provide a standardized platform for developing and deploying scalable enterprise systems. As a result of this, application servers are a core component of an enterprise system and an integral part of a new trend towards building service-oriented architectures. Today, the *three-tier architecture* paradigm has become an industry standard for building scalable client-server applications.

In multi-tier systems, frequent calls to application servers and databases place a heavy load on resources and may cause throughput bottlenecks and high server-side processing latency. Typically, preliminary system capacity estimates are done by using synthetic workloads or benchmarks which are created to reflect a "typical application behavior" for "typical client requests". While capacity planning based on synthetic workloads or benchmarks can be useful at the initial stages of design and development of a future system, it may not be adequate for answering more specific questions about an existing production system. Often, a service provider does need to answer the following questions:

- How many additional clients can be supported by the existing system *i)* while still providing the same performance guarantees, e.g., response time under 8 sec., and *ii)* assuming that new clients perform similar activities as already existing clients in the system, i.e., the system processes the same type of workload?
- If the client activities and behaviors change over time in a specified way, how is the performance of the system affected?

In this work, we propose a new capacity planning framework, called *R-Capriccio*, for practical capacity evaluation of existing production systems under "live" workloads that can provide answers to all of the above questions. *R-Capriccio* can assist in providing answers for advanced "what-if" scenarios in system capacity analysis where the evaluated system operates under a diverse workload mix. *R-Capriccio* is comprised of the following key components:

- *Workload profiler*: The profiler extracts a set of most popular client transactions, called *core* transactions, to characterize the overall site workload and the most popular client sessions at the site.
- *Regression-based solver*: Using statistical regression, the solver approximates the resource cost (CPU demand) of each core transaction on a given hardware. Thus a real workload mix can be directly mapped into the corresponding CPU demand requirements.
- *Analytical model*: For capacity planning of multi-tier applications with session-based workloads, an analytic model based on a network of queues is developed, where each queue represents a tier of the application.

Another important problem that needs to be addressed is a preliminary analysis of performance issues that often occur during the application updates and new software releases: this is also known as *anomaly detection*. Typically, when a new software release is introduced and unexpected performance problems are observed, it is important to separate performance issues that are caused by a high load of incoming workload from the performance issues caused by possible errors or inefficiencies in the upgraded software. R-Capriccio can be used to distinguish the performance issues that are not caused by the existing system workload and essentially be used as an alarm to identify anomalies in the system operation.

For most production multi-tier services the I/O traffic (both network and disk) is not a system bottleneck. The memory requirements increase linearly with the number of concurrent users in the system [2] and can be computed in a straightforward way. In this work, we concentrate on systems with CPU bottlenecks and evaluate the capacity requirements for support of a given workload with a specified constraint on the latency of user response times. This additional latency constraint makes this modeling problem non-trivial and challenging.

A prerequisite for applying our framework is that a service provider collects the following information:

- the application server access log that reflects all processed client requests and client activities at the site, and
- CPU utilization at all tiers of the evaluated system.

Thus the *problem* is to approximate the *CPU costs* of different client transactions at different tiers, and then use these cost functions to evaluate the resource requirement of scaled or modified transaction workload mix in order to accurately size the future system. In this work, we continue developing the approach that is based on linear regression for approximating the CPU transaction cost in a system running the TPC-W benchmark [24]. However, it is much more challenging to apply and validate this modeling approach with real, live workloads that exhibit much more complex and diverse behavior than the synthetic TPC-W benchmark.

To validate our approach, we use a 1-month long access logs and CPU utilization data from two heterogeneous application servers that provide customized client access to a popular and actively used HP service: Open View Service Desk (OVSD). We demonstrate that the proposed regression method provides a simple, but powerful solution to accurately approximate CPU transaction costs for both heterogeneous application servers under study. We use the results of the regression method to parameterize an analytic model of queues. We then use the analytic model to complete the last step of the capacity planning process and derive the maximum number of clients that the studied application servers can support for a given workload mix under different constraints on transaction response times.

The rest of the paper is organized as follows. Section 2 provides a detailed workload analysis and a workload profiler. Section 3 introduces our regression-based method for deriving the CPU cost of the site transactions. Section 4 presents the analytic model for predicting multi-tier application performance. Section 5 presents related work. Finally, a summary and conclusions are given in Section 6.

2 Workload Characterization

In this section, we analyze a 1-month trace collected from the heterogeneous application servers at the OVSD business portal during July 2006. This trace has a detailed information about each processed request, including its arrival and departure time, request URL, and client session ID.

2.1 Units of Client/Server Activities

Since often service providers are interested in capacity planning rules for their production systems under live, real workloads, we need to understand properties of these workloads, and identify a set of workload characteristics that are essential for a capacity planning framework.

We first define *client activity* as follows. Typically, a client communicates with a web service (deployed as a multi-tier application) via a web interface, where the unit of activity at the client-side corresponds to a download of a web page. In general, a web page is composed of an HTML file and embedded objects such as images. Typically, the HTML page is dynamically generated by the application server, and depending on the application and its business logic, the page generation may involve issuing multiple (or none) database calls. A browser retrieves a web page by issuing a series of HTTP requests for all objects: first it retrieves the main HTML file and after parsing it, the browser retrieves all the embedded images. Thus, at the server side, a web page retrieval corresponds to processing of multiple smaller objects that can be retrieved either in sequence or via multiple concurrent connections. It is common that a web server and application server reside on the same hardware, and shared resources are used by the application and web servers to generate main HTML files as well as to retrieve page embedded objects[1]. In the access logs that we obtained from the OVSD application server, there are both types of entries: web page requests and subsequent entries for embedded images. The HTTP protocol does not provide any means to delimit the beginning or the end of a web page: this is why it is very difficult to accurately measure the aggregate resources consumed due to web page processing at the server side. In this work, we define a *transaction* as a *web page* accessed by the client (also called *web page views*).

[1] It is common for applications in many production systems implemented using the PHP web-scripting/application development language [15].

Client access to a web service occurs in the form of a *session* consisting of multiple individual transactions (web pages). For example, in an e-commerce site, placing an order through the web site involves further requests relating to selecting a product, providing shipping information, arranging payment agreement, and finally receiving a confirmation. Thus, for a customer trying to place an order, or a retailer trying to make a sale, the real measure of such a web service performance is its ability to process the entire sequence of individual transactions needed to complete a higher-level logical transaction. The number of such concurrent client sessions that a multi-tier system can support without violating transaction response time is a measure of system capacity.

In this section, we present the analysis of OVSD workload performed by our *Workload Profiler*:

– first, it characterizes a set of client transactions and extracts the distribution of transactions over time;
– second, it characterizes a set of user activities by analyzing and extracting the session characteristics over time.

2.2 Transactions

In our analysis, we consider a reduced trace that contains only transactions (web page views) as discussed above. We omit all embedded images, style sheets, and other format-related primitives. Moreover, we further distinguish a set of unique *transaction types* and a set of client accesses to them. For static web pages, the URL uniquely defines a file accessed by clients. For dynamic pages the requests from different users to the same web page URL may appear as requests to different URLs due to the client-specific extension or a corresponding parameter list. We carefully filter out these client-specific extensions in the reduced trace.

There are 984,505 transactions in the reduced trace. Fig. 1 illustrates the number of transactions in each hour. It reflects a typical enterprise diurnal access pattern, i.e., high loads during work hours, and low loads during nights and weekends. In addition, the studied workload exhibits a regular and predictable load pattern.

Overall, in the reduced trace, there are 756 different unique transactions (or transaction types). Fig. 2 shows the cumulative distribution function (CDF) of client accesses to different transaction types ranked by the transaction popularity. The transaction with *rank 1* represents the most popular transaction type. Fig. 2 reflects that the studied workload exhibits a very high degree of reference locality: i.e., a small subset of site transactions is responsible for a very high percentage of client accesses, e.g.,

– the top 10 transaction types accumulate 79.1% of all the client accesses;
– the top 20 transaction types are responsible for 93.6% of the site accesses;
– the top 100 transaction types account for 99.8% of all site accesses.

This characterization is consistent with earlier works [5,6,7] that have demonstrated that web server and e-commerce workloads exhibit a high degree of reference locality. Complementary to the characterization of the most frequently accessed files, we also see that the percentage of the files that are requested only a few times over an entire month is very high for this site. These rarely accessed files may play a less important role in the capacity planning framework, as we demonstrate later.

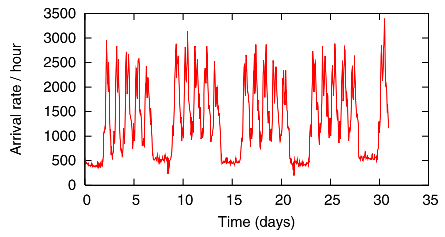


Fig. 1. Arrival rate of transactions for each hour in July, 2006

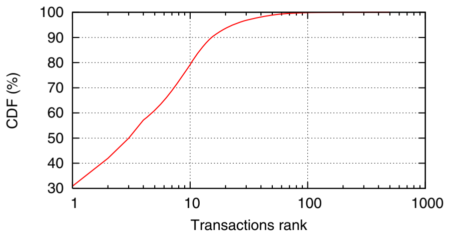
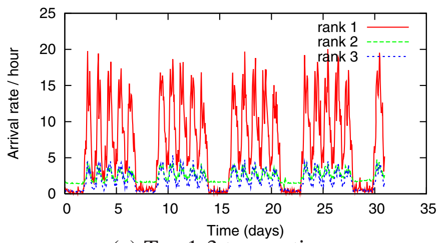
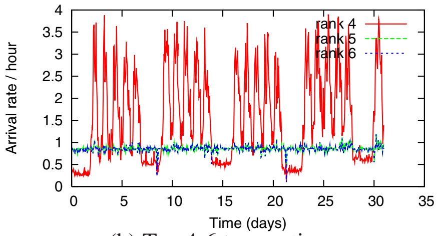


Fig. 2. CDF of the transaction types

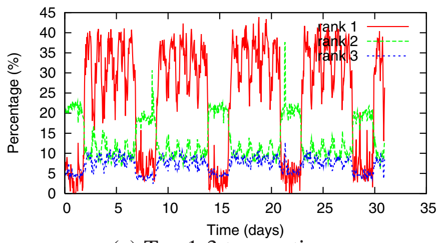


(a) Top 1-3 transactions

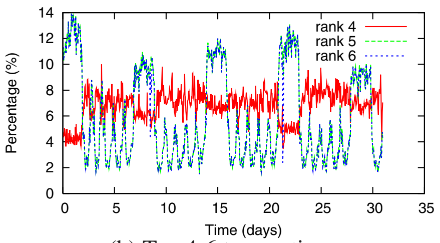


(b) Top 4-6 transactions

Fig. 3. Arrival rate of the first 6 most popular transactions across time



(a) Top 1-3 transactions



(b) Top 4-6 transactions

Fig. 4. Portions of the transactions belonging to the top 6 popular transactions across time

Fig. 3 shows the arrival rates of the transactions for the 6 most popular types over time, and Fig. 4 shows the percentages of these transaction types in the workload mix over time. Each point in these figures corresponds to one-hour statistics. The figure shows that the transaction mix is not stationary over time. For example, the most popular, rank 1 transaction can cotribute to 15% to 40% in the workload depending on the hour of the day. Similar observations apply to other transactions as well.

Traditional capacity planning methodologies usually examine peak loads and system utilization to conclude on the number of clients that can be handled by the system. These methods aim to accommodate variations in load while assuming that the set of workload transactions is stationary, i.e., that the distribution of different transaction types is fixed. Many of industry standard benchmarks are built using this principle [3,4]. But real workloads rarely exhibit this feature as shown by the analysis above. Therefore, instead

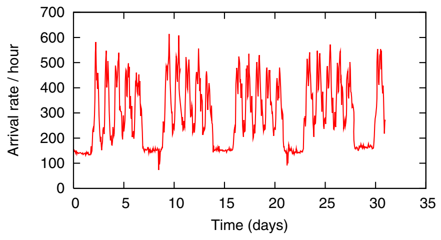


Fig. 5. Arrival rate of sessions for each hour in July, 2006

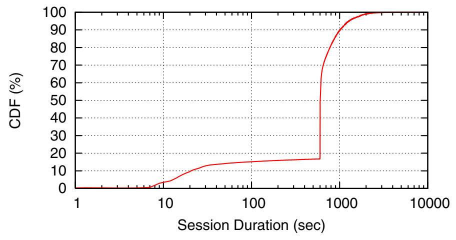


Fig. 6. CDF of the session durations

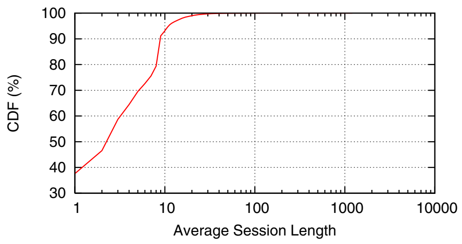


Fig. 7. CDF of the session length

of focusing on loads solely, a robust capacity planning methodology must also consider the changing workload mix since the system capacity directly depends on the types of user activities.

2.3 Sessions

Understanding user activities at the session level is essential for capacity planning, as *the number of concurrent sessions* in the trace is actually a representation of *the number of concurrent clients* handled by the system. Fig. 5 displays the arrival rate of new sessions over time, which follows the same trends as the transaction arrivals. Additionally, it indicates that the high load of transactions during peak time is mainly due to the increased number of customers.

Fig. 6 shows the CDF of client session durations. A session duration is defined as the time between the beginning of the first transaction and the end of the last transaction with the same session ID. The most typical session duration is around 600 seconds. It is related to the *timeout* parameter in the application server: if a session is inactive for 600 seconds it is timed out by the server.

Fig. 7 gives the CDF of the session length, i.e., the number of transactions within each session. Most sessions have a small number of transactions, i.e., 93.1% of the sessions have less than 10 transactions, and 37.6% of the sessions have only one transaction.

Since the traces are collected independently at two application servers supported by heterogeneous machines with different CPU speeds, we turn to the workload in each server to further understand the session-based behavior of users.

2.4 Workloads of Different Servers

In this sub-section, we present the workload and utilization analysis of each of the two application servers, which then is used by our capacity planning framework to show that the framework can effectively support heterogeneous resources.

The two application servers handle client requests after a load balancing point. Fig. 8 shows that the load balancing in this system works well. A similar number of transactions are dispatched to each of the two servers, and both exhibit the characteristics of the entire workload as described above. [2] Server 2 has a faster CPU. As a result, its CPU utilization is lower compared to server 1 (see Fig. 9). Most of the time, CPU utilization in both servers is under 10%. Note that for each weekend, there is a spike of CPU utilization which is related to administrator back-up tasks.

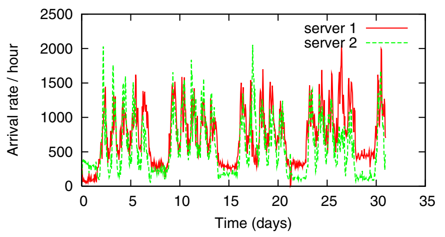


Fig. 8. Arrival rate of transactions of each application server

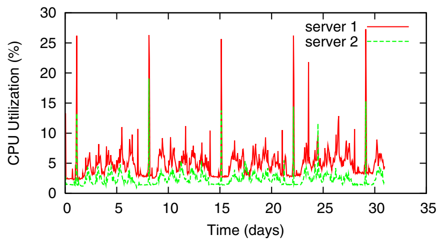


Fig. 9. Average CPU utilization of each application server

Fig. 10 shows the average number of concurrent sessions over time processed separately by server 1 and by server 2. During peak time, there are about 60 concurrent sessions for each server, but during the weekends, the number of concurrent sessions decreases to 10.

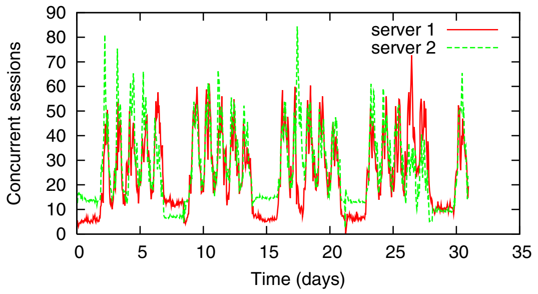


Fig. 10. Average number of concurrent sessions of each application server

[2] The workload mixes and the transaction popularity ranking at each server are similar to the entire system. We do not report the figures here due to space limitation.

When server 2 receives a slightly higher number of requests than server 1 (since server 2 has a faster CPU, and its typical CPU utilization is lower), this leads to a slightly higher number of concurrent sessions hosted by the server 2 as shown in Fig. 10.

2.5 Summary of Workload Analysis

To summarize, the following observations have to be taken into account for an accurate capacity planning and performance evaluation of production systems with live workloads:

- The transaction mix varies over time and hence can not be treated as a fixed, stationary distribution.
- The workloads exhibit a strong locality property, i.e., a small number of transaction types are responsible for a large fraction of client requests.
- Most of users have a high think time.

The *Workload Profiler* collects a set of the following metrics over time: *i)* the average CPU utilization, *ii)* the number of different transactions, *iii)* the number of concurrent sessions, and *iv)* the client think times. These metrics are collected for each time window of 1 hour (this is a tunable tool parameter) and for each application server. These metrics can then be used to parameterize the analytic model in Section 4.

3 CPU Cost of Transactions

In this section, we use a statistical regression-based approach for an efficient approximation of CPU demands of different transaction types. We have introduced this approach in our earlier paper [24], where we evaluated it by using a testbed of a multi-tier e-commerce site that simulates the operation of an on-line bookstore, according to the classic TPC-W benchmark [4]. The challenge is to apply and validate this technique with real, live workloads that exhibit much more complex and diverse behavior than synthetic ones. With the knowledge of CPU demands of transactions one can easily compose the resource requirement of scaled or modified transaction mixes. Thus, this methodology can be directly applied to production systems and can be used to explain large-scale system behavior and predict future system performance. In this section, we analyze challenges of applying this method to production systems operating under live, real workloads, and introduce an optimization technique that enables an efficient use of the proposed approach.

3.1 Regression Methodology

To capture the changes in server workload we observe a number of different transactions over fixed length time intervals, denoted as *monitoring windows*. The transaction mix and system utilization are recorded at the end of each monitoring window.

Assuming that there are totally M transaction types processed by the server, we use the following notations:

- T is the length of the monitoring window;
- N_i is the number of transactions of the i-th type, where $1 \leq i \leq M$;
- $U_{CPU,n}$ is the average CPU utilization at the n-tier during this monitoring window;
- $D_{i,n}$ is the average service time of transactions of the i-th type, at the n-tier of the systems, where $1 \leq i \leq M$.
- $D_{0,n}$ is the average CPU overhead related to activities that "keep the system up". There are operating system processes or background jobs that consume CPU time even when there is no transaction in the system.

From the utilization law, one can easily obtain Eq. (1) for each monitoring window [8]:

$$D_{0,n} + \sum_i N_i \cdot D_{i,n} = U_{CPU,n} \cdot T. \tag{1}$$

Because it is practically infeasible to get accurate service times $D_{i,n}$ (since it is an over-constrained problem), we let $C_{i,n}$ denote the approximated CPU cost of $D_{i,n}$ for $0 \leq i \leq M$. Then an approximated utilization $U'_{CPU,n}$ can be calculated as

$$U'_{CPU,n} = \frac{C_{0,n} + \sum_i N_i \cdot C_{i,n}}{T}. \tag{2}$$

To solve for $C_{i,n}$, one can choose a regression method from a variety of known methods in the literature. Finding the best fitting method is outside of the scope of this paper. In all experiments, we use the Non-negative Least Squares Regression (Non-negative LSQ) provided by MATLAB to get $C_{i,n}$. This non-negative LSQ regression minimizes the error

$$\epsilon = \sqrt{\sum_j (U'_{CPU,n} - U_{CPU,n})_j^2} \quad ,$$

such that $C_{i,n} \geq 0$, where j is the index of the monitoring window over time.

3.2 Applying Regression to a Production System with Live Workload

We use the one-month trace analyzed in Section 2 to evaluate the accuracy of the regression-based method described above. We had to limit our validation exercise to the application server tier because we could not get relevant CPU utilization measurements at the database tier.

For each 1-hour time window[3] the *Workload Profiler* provides the average CPU utilization as well as the number of transactions N_i for the i-th transaction type, where $1 \leq i \leq M$. The OVSD trace profile has the format shown in Table 1.

When we first introduced and applied the regression-based technique for evaluating the transaction cost in [24], there were only 14 different transaction types in TPC-W. The analysis of OVSD workload revealed that the real workloads often have a much higher number of transaction types, e.g., OVSD workload operates over 756 different transaction types. In order to apply the regression technique to OVSD workload we

[3] In [24], we showed that a larger monitoring window improves the accuracy of regression results. For the production system under study a monitoring window of 1 hour produced the best results.

Table 1. An example of transaction profile in server 1

Time (hour)	N_1	N_2	N_3	N_4	$\cdots$	N_{756}	$U_{CPU}(\%)$
1	21	15	21	16	$\cdots$	0	13.3201
2	24	6	8	5	$\cdots$	0	8.4306
3	18	2	5	4	$\cdots$	0	7.4107
4	22	2	4	7	$\cdots$	0	6.4274
5	38	5	6	7	$\cdots$	0	7.5458
$\cdots$							

would need to collect more than 756 samples of 1-hour measurements. Such a collection would require to observe this workload for more than 1-month before we would collect enough "equations" for evaluating the OVSD transaction cost.

The workload analysis presented in Section 2.2 shows that the studied workload exhibits a very high degree of reference locality, i.e., a small subset of site transactions is responsible for a very high percentage of client accesses, e.g., the 100 most popular transactions already cover 99.8% of all client accesses. From the other side, there is a high percentage of transactions that are rarely accessed, i.e., so called, "one-timers". We divided the original 1-month trace in two halves. The additional workload analysis revealed that there are 203 transactions that are accessed only once in the first half of the trace, and which are not accessed in the second half of the trace. Similarly, there are 189 transactions that are accessed only once in the second half of the trace, and which are not accessed in the first half of the trace. The non-negative LSQ regression used in this paper returns "0" as a typical value for "rare" variables, since there is not enough information in the original set of equations to produce a more accurate solution.

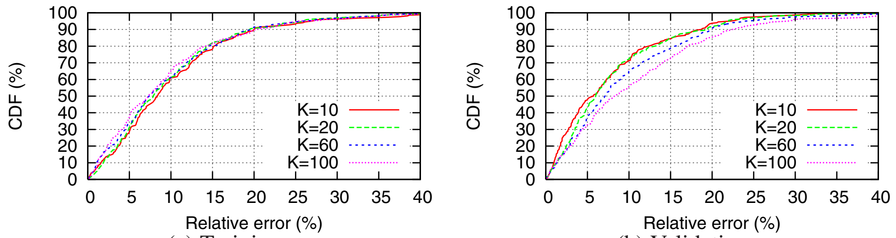
So, the question is whether accurate performance results can be obtained by approximating the CPU cost of a much smaller set of popular (*core*) transactions. In other words, if we use regression to find the CPU cost of a small number of *core* transactions, can this small set be useful for an accurate evaluation of the future CPU demands in the system?

Following this idea, we only use the columns N_1 to N_K and U_{CPU} in Table 1 to approximate C_i for $1 \leq i \leq K$. The approximated U'_{CPU} of every hour is then computed by these N_1 to N_K and C_1 to C_K values.

We also consider the results produced by the non-negative LSQ regression method when K is equal to 10, 20, 60 and 100 transactions respectively. We use the relative error of the approximated utilization as the metric to validate the regression accuracy. For every hour, the relative error of the approximated utilization is defined as

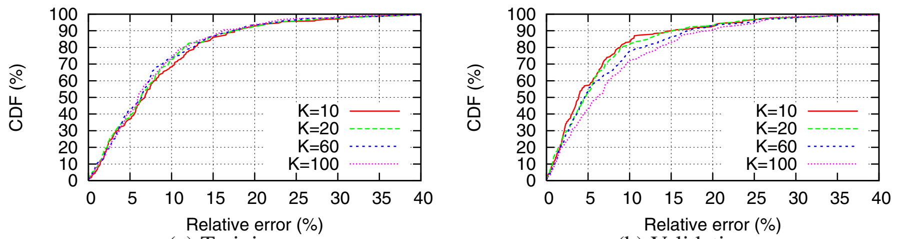
$$Error_R = \frac{|U'_{\text{CPU}} - U_{\text{CPU}}|}{U_{\text{CPU}}}. \tag{3}$$

We divide the OVSD trace into two parts. The first half is used as a training set to solve for the CPU cost C_i using the non-negative LSQ regression method. The second half is treated as a validation set. Because the administration jobs during weekends might introduce a significant noise to the CPU utilization, the training set for the regression consists of data from workdays only.



(a) Training (b) Validating

Fig. 11. Server 1. CDF of relative errors under a different number of of core transactions chosen for a regression method: (a) training set, (b) validating set.



(a) Training (b) Validating

Fig. 12. Server 2. CDF of relative errors under a different number of core transactions chosen for a regression method: (a) training set, (b) validating set.

The regression method produces similar results for the two heterogeneous application servers in the system. Figs. 11-12 show the CDF of the relative errors for training and validating sets for servers 1 and 2, respectively.

The regression results can be summarized as follows:

- Overall, the non-negative LSQ regression achieves good results for all examined values of K, i.e., when the regression method is applied to approximate the CPU cost of the top 10, 20, 60, or 100 most popular transactions. For the training set, at least 60% of the points have relative errors less than 10%, and at least 90% of the points have relative errors less than 20% (see Figs. 11(a) and 12(a)). The method's accuracy for the validating set is only slightly worse (see Fig.11(b), 12(b)).
- Larger K achieves a higher accuracy for the training set. However, this improvement is not significant: for $K = 100$ there is only a 4% improvement compared to the results with the top 10 transactions.
- The larger values of K, e.g., $K = 100$, show a worse prediction accuracy for the validating set compared to K equal to 10 or 20 core transactions as shown in Fig. 11 - 12. These results again can be explained by the workload properties. While we consider 100 most popular transactions, the last 80 of them only responsible for 6% of the client requests. These transactions have an irregular access pattern. Some of those transactions appear only in the first or second half of the trace

(while not being a "one-timer"). As a result, computing the individual cost of these transactions does not help to evaluate the future CPU demands, and introduces a higher error compared to the regression based on a smaller transaction set.

Regression produces the best results when a representative set of core transactions is used and rarely accessed transactions are omitted. Since some of the rarely accessed transactions might only appear in the first half of the trace, while some different rarely accessed transactions may only appear in the second half of the trace, it is beneficial to use only core transactions in linear regression as well as in the overall capacity planning. The additional CPU overhead that is due to the rarely accessed transactions is "absorbed" by the CPU cost of the core transactions. Consequently, a small additional CPU usage by the distinct and rarely accessed transactions is accounted via the CPU cost of the most frequently and consistently accessed core transactions.

We conclude that considering the top 20 *core* transactions (i.e., $K = 20$) leads to the most accurate results. Note that the top 20 transactions are responsible for 93.6% of the total transactions in the analyzed trace. Therefore, selecting the top K transactions that account for 90% - 95% of all client accesses for the regression method results in a good representative subset of the entire workload. The regression solver produces a solution for 200 equations with 20 variables only in 8 millisecond. In general, the common least squares algorithms have polynomial time complexity as $O(u^3v)$ when solving v equations with u variables, and hence, can be efficiently used as a part of online resource evaluation method [1]. Combining the knowledge of workload properties with statistical regression provides a powerful solution for performance evaluation of complex production systems with real workloads.

3.3 Anomaly Detection

Shortened product development cycle, frequent software updates, and more complex integration dramatically increase the risk of introducing poorly performing applications. Consequently, another problem that needs to be addressed is a preliminary analysis of performance issues that often occur during the application updates and new software releases: this is also known as *anomaly detection*. Typically, when a new software release is introduced and unexpected performance issues are observed, it is important to make sure that these performance issues are not caused by the current workload, i.e., system overload due to a higher rate of client requests. When the system performance can not be explained by the existing workload mix in the system, it suggests that the observed performance issues might be caused by the latest software modification. Thus, it is important to evaluate the resource usage caused by the existing transaction mix in the system, and to generate the alarm events when system utilization significantly deviates from the predicted utilization value computed from the existing workload.

Using the observed workload mix we compute the expected CPU utilization of the system U'_{CPU} by Eq. 2 and compare it against the measured CPU utilization U_{CPU} for the same time period. The service provider can set a threshold Th that defines the acceptable deviation of expected system utilization U'_{CPU} from the observed utilization U_{CPU}. If

$$\frac{U_{CPU} - U'_{CPU}}{U'_{CPU}} \geq Th \qquad (4)$$

then our tool generates an alarm event. We only consider the situations when the measured CPU utilization is significantly higher than the expected one, since in this case, something else besides the observed workload causes performance problems.

Fig. 13 demonstrates the anomaly detection feature of the tool for the OVSD trace with $Th = 2$. Our method accurately predicts CPU utilization caused by this mix. Over weekends our method has generated the alarm warnings (marked with circles in Fig. 13) indicating that something else, besides the transaction processing, happens in the system. During these time intervals the predicted and observed utilizations are drastically different. Our method correctly identifies a non-typical CPU utilization caused by a set of additional administrative tasks, extensively performed over weekends (see remarks about this in Section 2.4), and which had nothing to do with the processed transaction mix.

While in this paper, we defined an anomaly situation as one where observed CPU utilization significantly exceeds predicted CPU utilization, one can consider a symmetrical situation where observed CPU utilization is significantly lower than predicted CPU utilization as a result of transaction mix, and verify the reasons behind it: for example, it might be related to unavailable embedded objects in the serviced web pages due to some storage subsystem problems. Currently, we are working on optimizing the regression technique that provides a better support for performance anomaly detection as well as on designing a technique for tuning the threshold parameters that minimize false positive alarms.

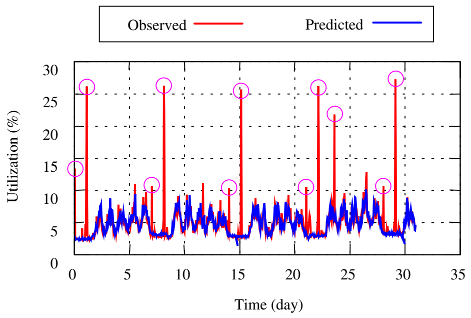


Fig. 13. Anomaly detection with R-Capriccio

4 Capacity Planning

Modern Internet servers typically employ a multi-tier structure consisting of web servers, application servers and databases as given in Fig. 14. Each tier gets the requests from its preceding tier, and may generate certain requests to its successor. For scalability, a tier may consist of several replicated servers. These servers may be heterogeneous, and a dispatcher may employ a special load balancing strategy for distributing the incoming requests across the replicated servers.

Due to the session-based client behavior, a multi-tier system is usually modeled as a closed system with a network of queues (see Fig. 15). The number of clients in the system is fixed. When a client receives the response from the server, it issues another

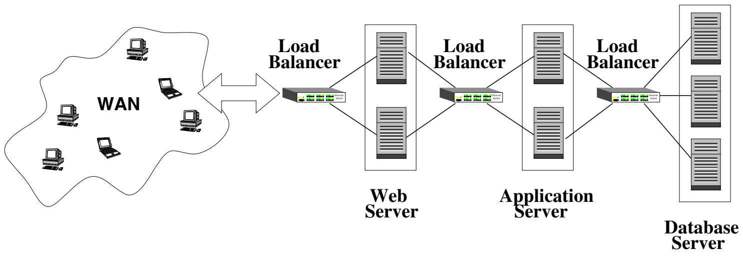


Fig. 14. A multi-tier structure of a server

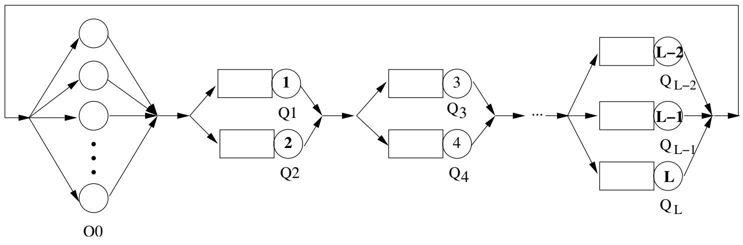


Fig. 15. Queuing network modeling of a multi-tier closed system

request after certain think time. This think time is modeled as an infinite server Q_0 in Fig. 15. Once the service time in each queue is obtained, this closed system can be solved efficiently using Mean-Value Analysis (MVA) [8].

Workload characterization of real traces in Section 2 shows that the workload mix changes over time, and hence the service time could not be modeled as a fixed distribution for the entire lifetime of the system but one can treat the workload as fixed during shorter time intervals (e.g., 1 hour). R-Capriccio performs the capacity planning procedure for each monitoring time window of 1 hour and then combines the results across these time points to get the overall solution [4].

4.1 MVA

MVA is based on the key assumption that when a new request enters a queue, this request sees the same average system statistics in the system as without this new request. Fig. 16 presents a description of the detailed MVA algorithm [22].

The visit ratio V_i (definition in Fig. 16) is controlled by the load balancing policy. For example, if the load balancing policy used is equally partitioning the transactions across all servers, then the number of visits V_s to server s in tier l is equal to $1/m_l$, where m_l is the number of servers in tier l.

[4] For the TPC-W benchmark and most production multi-tier services CPU is a typical system bottleneck. However, in practice, when one needs to make a projection of the maximum achievable system throughput, additional "back of the envelope" computations for estimating memory and network requirements under the maximum number of concurrent clients are required to justify this maximum throughput projection.

```
Inputs:
$N$ = number of clients
$Z$ = think time
$L$ = number of servers
$S_i$ = service time per visit to the $i$-th queue
$V_i$ = number of visits to the $i$-th queue

Outputs:
$X$ = system throughput
$Q_i$ = average number of jobs at the $i$-th queue
$R_i$ = average response time of the $i$-th queue
$R$ = system response time (excluding think time)
$U_i$ = utilization of the $i$-th queue

1. Initialization: for $i = 1$ to $L$ do $Q_i \leftarrow 0$
2. Iterations:
   for $n = 1$ to $N$ do
   a. for $i = 1$ to $L$ do
          $R_i = S_i(1 + Q_i)$
   b. $R = \sum_{i=1}^{L} R_i V_i$
   c. $X = \frac{n}{Z + R}$
   d. for $i = 1$ to $L$ do $Q_i = X V_i R_i$
3. for $n = 1$ to $L$ do
   a. $X_i = X V_i$
   b. $U_i = X S_i V_i$
```

Fig. 16. The MVA algorithm [8]

Note that the original MVA (as in Fig. 16) takes the number of clients N as input, and computes the average performance metrics for a system with N clients. In capacity planning, the number of clients is unknown. In the contrary, the model needs to be solved for exactly this unknown variable. Here, we assume that the Service Level Agreement (SLA) specifies a threshold $\varGamma_R$ (i.e., upper bound) of the average transaction response time. Then the condition in step **2** of MVA is changed to the following condition: "while $R \leq \varGamma_R$ do".

4.2 Case Study

In this section, we demonstrate how *R-Capriccio* helps to answer the following capacity planning question:

– How many clients can be supported by the existing system:
  - providing the desirable performance guarantees, e.g., response time under $\varGamma_R$, and
  - assuming that the system processes a given (varying, non-stationary) type of workload?

The detailed sequence of steps performed by *R-Capriccio* is summarized in Fig. 17.

```
1. Workload profiler:
   For each monitoring window $w$, and each server $s$:
   a. collect the number of transactions $N_{i,s,w}$ of each type $i$;
   b. collect the average utilization $U_{s,w}$.
   For each monitoring window $w$:
   a. select the top $K$ most popular transaction types;
   b. collect the transaction mix in the system,
      i.e., the percentage $p_i$ of the transactions for type $i$
      for all $1 \leq i \leq K$;
   c. collect the average think time $Z_w$.

2. Regression-based solver:
   For each server $s$:
      Compute the cost function $C_{i,s}$ for each transaction
      type $i$ as described in Section 3, where $1 \leq i \leq K$.

3. Analytical model:
   For each monitoring window $w$:
   a. approximate the service time $S_s$ for each server $s$ as
      $S_s = \sum_{i=1}^{K} p_i \cdot C_{i,s}$;
   b. compute the maximum number of clients $MAX_w$ can
      be handled with average response time less than $\mathit{\Gamma}_R$
      using MVA algorithm.
   Build the profile with entry as $(w, MAX_w)$.
   Find the minimal $MAX_w$ value $X$.
 $X$ is the number of concurrent customers the system can support
with the average transaction response time less than $\mathit{\Gamma}_R$
```

Fig. 17. The R-Capriccio Framework

The first two steps of R-Capriccio that use the *Workload Profiler* and the *Regression-based Solver* have been presented in the previous two sections. We use the same workload as input to the third step of the analytic model. In the case study, we had to limit our capacity planning exercise to the application server tier (which is a bottleneck tier in the OVSD service) because we could not get relevant CPU utilization measurements at the database tier (this particular database was shared across a few different services, and we had only access to the OVSD part of the application servers).

Since the traces are collected from the two servers independently, we treat each heterogeneous server as an independent system. Later, we show how to combine the capacity planning results from those heterogeneous servers together.

All the experiments are conducted for the top 20 most popular transaction types, i.e., K is set to 20. Following step **3.a.** in Fig. 17, we approximate the average service time for each 1-hour time interval for both servers as shown in Fig. 18. Because server 2 has a faster CPU, it is expected that it has a smaller service time than server 1. For each time interval there is a vector of parameters representing the average think time, the average

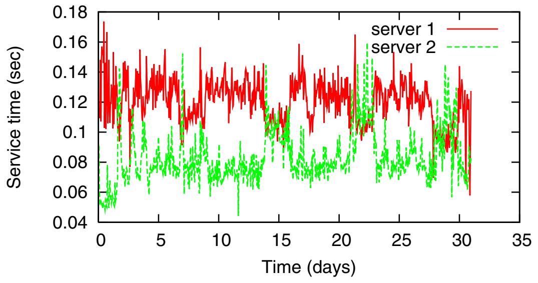


Fig. 18. Approximated service time using the CPU cost of the top 20 transaction types

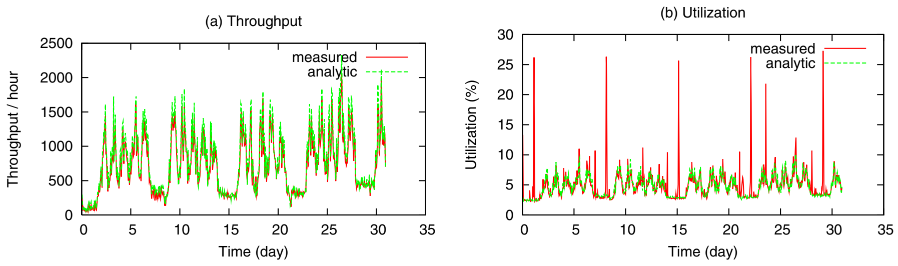


Fig. 19. Server 1. Measurements versus analytic model: a) Throughput of transactions; b) CPU utilization.

service time, and the number of concurrent clients. We apply the MVA model at each time interval for each server.

Fig. 19(a) shows the validation results by comparing the throughput of the analytic model and the measured transaction throughput of server 1. The analytic model captures the real system behavior well, i.e., 90% of the relative errors are below 18.7%. Copmarisons of the throughput of the analytic model and the measured session throughput of server 2 are of similar accuracy.

Fig. 19(b) compares the average measured utilization over time with the utilization results provided by the analytic model. We observe a nearly perfect match between the measured and analytic results. Except for the utilization spikes observed in the real system measurements over weekends that are due to special administration-related tasks as discussed in Sections 2.4 and 3.3. Our method predicts a much lower CPU utilization using the observed transaction mix for these time periods. This presents an additional functionality of R-Capriccio that can help in generating "alarm" conditions when predicted utilization for processing the existing workload significantly deviates from the system measurements. The analytic results for server 2 show a similar performance trends and are not presented here for brevity.

Fig. 20 and Fig. 21 illustrate the CDF of the maximum number of clients that can be supported by server 1 and server 2 under the changing OVSD transaction mix over time, where the transaction response time is limited by Γ_R equal to 1, 3, 6 and 10 seconds respectively. These results are computed using the same think time and service time as in the above experiments.

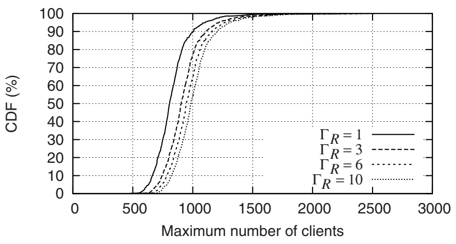


Fig. 20. Server 1: CDF of the maximum number of clients under different threshold Γ_R of the average response time

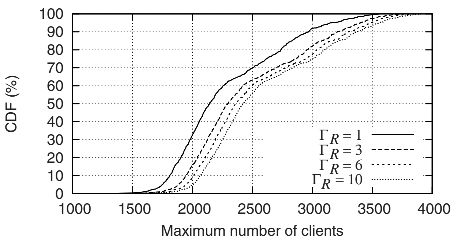


Fig. 21. Server 2: CDF of the maximum number of clients under different threshold Γ_R of the average response time

The summary of results are shown in Table 2. As expected, server 2 has a much higher capacity than server 1. Higher values in threshold Γ_R allow for a larger number of clients to be supported by the system.

Table 2. Maximum number of clients under different Γ_R

Γ_R(sec)	Server 1	Server 2	Total
1	472	1349	1821
3	528	1478	2006
6	565	1534	2099
10	608	1580	2188

The capacity of the entire application server composed of these two heterogeneous servers is determined by the load balancing policy as well. For example, if the SLA defines that the average transaction response time is not higher than 1 second, the studied application server can handle 1821 concurrent clients but only if the load balancer is aware of the heterogeneous capacity of these two servers and can split the load proportionally to server capacity. If the load balancer partitions transactions equally, capacity reduces to 944, just half of the previous one. Such a big difference indicates the significant impact of a load balancing policy on system capacity as heterogeneous CPU speeds must be taken into account.

5 Related Work

Performance evaluation and capacity planning of software and hardware systems is a critical part of the system design process [8]. There is a number of capacity planning techniques proposed for different popular applications.

Among these techniques, queuing theory is a widely used methodology for modeling a system behavior and answering capacity questions [16,17,18]. Modeling of a single-tier system, such as a simple HTTP server, has been studied extensively. Even for a multi-tier structure which is employed ubiquitously for most servers, the system is usually abstracted as the most bottle-necked tier only: in [16], only the application tier for the e-commerce systems are modeled by a M/GI/1/PS queue; similarly in [19] the application tier with N node cluster is modeled by a G/G/N queue. Recently B. Urgaonkar et al. proposed analytic models for both open and closed multi-tier systems [17,18]. These models are validated by synthetic workloads running in real systems. However the expense of accurately estimating model parameters, i.e., service times and visit ratios, from each server log makes this model difficult to apply in production environments. Direct measurements in [18] do not characterize transactions as we do in this paper. Moreover, existing capacity planning methods are based on evaluating the system capacity for a fixed set of typical user behaviors. Once the service time is estimated, it is consistent throughout the planning procedure. This approach does not consider the fact that a changing workload for the same system has different service times and may result in different system capacity. Our experiments show that such techniques as those in [18] may fail to model a real system because of its dynamic nature.

In this paper, we use a similar closed multi-tier model as in [18], but in contrast to [18] or other examples in the existing literature of capacity planning, we propose a methodology that does not need a controlled environment for analytic model parameterization. Instead of characterizing the overall service time of every server, we use a statistical regression method to approximate the service cost of individual transactions. This CPU cost function together with the transaction mix help to approximate the system service time that varies with the changing transaction mix.

The use of statistical methods in capacity planning has been proposed in the early 80's [9,8], but the focus was on a single machine/cluster that is much simpler than current large-scaled multi-tiered systems. Recently statistical methods are getting more attention in computer performance analysis and system performance prediction. In [20] the authors use multiple linear regression techniques for estimating the mean service times of applications in a single-threaded software server. These service times are correlated with the Application Response Measurement package (ARM) data to predict system future performance. In [21],[23] the authors focus on transaction mix performance models. Based on the assumption that transaction response times mostly consist of service times rather than queueing times they use the transaction response time to approximate the transaction service demand. The authors use linear regression to identify performance anomalies in past workloads and to scrutinize their causes. We do not use measured transaction response times to derive CPU transaction demands (this approach is not applicable to the transactions that themselves might represent a collection of smaller objects). One of their basic assumptions is that the transaction mix consists of a small number of transaction types.

We have introduced a statistical regression-based approach for the CPU demand approximation of different transaction in our earlier paper [24], where we evaluated this approach by using a testbed of a multi-tier e-commerce site that simulates the operation of an on-line bookstore, according to the classic TPC-W benchmark [4]. Using the TPC-W benchmark, we demonstrated that the use of linear regression provides promising results. However, TPC-W operates using only 14 transaction types. In this work, we continue applying the linear regression technique for approximating the CPU transaction cost as was introduced in [24] but in a much more challenging environment. Here, we applied and validated this technique with real, live workloads that exhibit much more complex and diverse behavior than the synthetic TPC-W benchmark. Among the contribution of the current paper is a novel approach that illustrates how the regression-based technique can be applied to the production sites with large set of transaction types. By applying the regression to a set of popular, so-called "core" transactions (that are responsible for 90% - 96% of the site traffic) we are able to obtain the accurate estimates of transaction CPU cost that can be used for a variety of performance anomaly detection cases and capacity planning tasks in the production sites with real, live workloads.

6 Conclusion

In this paper, we present *R-Capriccio*, a new capacity planning framework which provides a practical, flexible and accurate toolbox for answering capacity planning and anomaly detection questions for multi-tier production systems with real workloads. More importantly, it can be used for explaining large-scale system behavior and predicting future system performance.

We used the access logs from the OVSD application servers to demonstrate and validate the three key components of R-Capriccio: the workload profiler, the regression-based solver, and the analytic model. In our capacity planning framework, we identify the set of most popular *core* transactions and sessions for building a site profile, compute transaction cost, and size the future system under the real workload. In order to derive the resource cost of each core transaction (i.e., CPU time required for corresponding transaction processing), we observe a number of different core transactions over fixed length time intervals and correlate these observations with measured server utilization for the same time interval. Using a *non-negative least-squares regression* method we approximate the resource cost of each core transaction. The statistical regression works very well for estimating the CPU demands of transactions that themselves might represent a collection of smaller objects and where the direct measurement methods are not feasible.

While this paper concentrates on evaluating the CPU capacity required for support of a given workload, we believe that regression methods can be efficiently applied for evaluating other shared system resources. We plan to exploit this avenue in our future work.

References

1. Ari, B., Giivenir, H.A.: Clustered Linear Regression. Knowledge-Based Systems 15(3) (2002)
2. Capacity Planning for WebLogic Portal, `http://edocs.bea.com/wlp/docs81/capacityplanning/capacityplanning.html`
3. The Workload for the SPECweb96 Benchmark, `http://www.specbench.org/osg/web96/workload.html`
4. TPC-W Benchmark, `http://www.tpc.org`
5. Arlitt, M., Williamson, C.: Web Server Workload Characterization: The Search for Invariants. In: Proc. of the ACM SIGMETRICS 1996 Conference, Philadelphia, PA (May 1996)
6. Almeida, V., Bestavros, A., Crovella, M., de Oliveira, A.: Characterizing Reference Locality in the WWW. Technical Report, Boston University, TR-96-11 (1996)

7. Arlitt, M., Krishnamurthy, D., Rolia, J.: Characterizing the Scalability of a Large Web-based Shopping System. J. ACM Transactions on Internet Technology 1(1) (August 2001)
8. Menasce, D., Almeida, V., Dowdy, L.: Capacity Planning and Performance Modeling: from mainframes to client-server systems. Prentice Hall, Englewood Cliffs (1994)
9. Kachigan, T.M.: A Multi-Dimensional Approach to Capacity Planning. In: Proc. of CMG Conference 1980, Boston, MA (1980)
10. Cherkasova, L., Tang, W.: Sizing the Streaming Media Cluster Solution for a Given Workload. In: CCGrid 2004. Proc. of the 4th IEEE/ACM, Chicago, USA (2004)
11. Rolia, J., Cherkasova, L., Arlitt, M., Andrzejak, A.: A Capacity Management Service for Resource Pools. In: Proc. of the Fifth Int. Workshop on Software and Performance (2005)
12. Chase, J.S., Anderson, D., Thakar, P., Vahdat, A., Doyle, R.: Managing Energy and Server Resources in Hosting Centers. In: SOSP. Proc. of the 18th ACM Symposium on Operating System Principles (2001)
13. Sarris, D., Hofer, J.: Capacity Planning for e-Commerce Systems With Benchmark Factory, www.dlt.com/quest/
14. Klerk, L., Bender, J.: Capacity Planning. Microsoft TechNet (2000), http://www.microsoft.com/technet/archive/itsolutions/ecommerce
15. PHP HyperText preprocessor, www.php.net
16. Villela, D., Pradhan, P., Rubenstein, D.: Provisioning Servers in the Application Tier for E-Commerce Systems. In: Proc. of IWQoS 2004, Montreal, Canada (2004)
17. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P.: Dynamic Provisioning of Multi-tier Internet Applications. In: ICAC 2005. In Proc. of the 2nd IEEE International Conference on Autonomic Computing, Seattle (June 2005)
18. Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., Tantawi, A.: An Analytical Model for Multi-tier Internet Services and its Applications. In: Proc. of the ACM SIGMETRICS'2005, Banff, Canada (June 2005)
19. Ranjan, S., Rolia, J., Fu, H., Knightly, E.: QoS-Driven Server Migration for Internet Data Centers. In: Proc. of IWQoS 2002, Miami, FL (May 2002)
20. Rolia, J., Vetland, V.: Correlating Resource Demand Information with ARM Data for Application Services. In: Proc. of the ACM Workshop on Software and Performance (1998)
21. Kelly, T.: Detecting Performance Anomalies in Global Applications. In: WORLDS 2005. Second Workshop on Real, Large Distributed Systems (2005)
22. Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for experimental Design, Measurement, Simulation, and Modeling. Wiley-Interscience, NY (1991)
23. Stewart, C., Kelly, T., Zhang, A.: Exploiting Nonstationarity for Performance Prediction. In: Proc. of EuroSys 2007, Lisbon, Portugal (March 2007)
24. Zhang, Q., Cherkasova, L., Smirni, E.: A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications. In: ICAC 2007. Proc. of the Fourth International Conference on Autonomic Computing, Jacksonville, FL, p. 27 (2007)