Web Transaction Analysis and Optimization (TAO)

Pankaj K. Garg, Ming Hao, Cipriano Santos,
Hsiu-Khuern Tang, Alex Zhang
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2002-45
March 4th , 2002*

E-mail: garg@hpl.hp.com

software performance monitoring, Web performance tuning, application response monitoring, software performance modeling

Web user interactions start at a client (a browser on an access device) and flow through several software components connected via digital networks. While the business world is fascinated with the range of possibilities for such E-transactions, the underlying technology opens up several new challenges for managing the performance, availability, and reliability of the enabling software components. Several studies have shown that poor response times for Web user interactions can significantly impact the business values--slow Web transactions can have strong implications on brand-name, customer acquisition and loyalty, and purchasing behavior of customers.

In the TAO project, we are developing metrics, models, and infrastructure to effectively manage the performance of Web applications. We use WebMon, a novel instrumentation tool to obtain transaction-level data from web interactions, from end-user and system component perspectives. Our analysis techniques help determine important segments of users and their web interactions. The analysis is embedded in visualization and optimization modules, enabling efficient reporting for system and business administrators, and automated resource scheduling and planning. In this paper we present an overview of the TAO project, and highlight some of its novel aspects, e.g., use of pixel-bar charts, web request classification, and integrated demand and capacity planning.

Web Transaction Analysis and Optimization (TAO)

Pankaj K. Garg, Ming Hao, Cipriano Santos,
Hsiu-Khuern Tang, and Alex Zhang

HP Labs
1501 Page Mill Road
Palo Alto, CA 94304
garg@hpl.hp.com

February 28, 2002

Abstract

Web user interactions start at a client (a browser on an access device) and flow through several software components connected via digital networks. While the business world is fascinated with the range of possibilities for such E-transactions, the underlying technology opens up several new challenges for managing the performance, availability, and reliability of the enabling software components. Several studies have shown that poor response times for Web user interactions can significantly impact the business values–slow Web transactions can have strong implications on brand-name, customer acquisition and loyalty, and purchasing behavior of customers.

In the TAO project, we are developing metrics, models, and infrastructure to effectively manage the performance of Web applications. We use WebMon, a novel instrumentation tool to obtain transaction-level data from web interactions, from end-user and system component perspectives. Our analysis techniques help determine important segments of users and their web interactions. The analysis is embedded in visualization and optimization modules, enabling efficient reporting for system and business administrators, and automated resource scheduling and planning. In this paper we present an overview of the TAO project, and highlight some of its novel aspects, e.g., use of pixel-bar charts, web request classification, and integrated demand and capacity planning.

Keywords: Software Performance Monitoring, Web Performance Tuning, Application Response Monitoring, Software Performance Modeling.

1 Introduction

Since its inception in the mid-90's, the World Wide Web (Web) [4] has become a ubiquitous information dissemination mechanism. In addition, the business community has found innovative use of the Web as a means of conducting retail, auction, and other e-transactions. The back-end information processing capabilities of web servers can also be invoked with the use of automated agents or computing clients, resulting in *e-services* or *web-services.* Such e-services can be orchestrated with a process model to support business processes over the Internet. In short, *the Web is a fundamental, new, pervasive architecture for conducting information processing transactions.*

A Web transaction starts at a client (browser, automated component, or appliance device) and flows through several computing and network devices on its way to a back-end application server, using the Hypertext Transfer Protocol [4]. The *response time* for a Web transaction is determined by the queueing and service times in the various components that a transaction flows through, and additional services that it has to wait for, e.g., DNS [12]. Some of these components, like, DNS, Internet routers and hubs, are not under the control of the Web service providers. Other components, e.g., the back-end web server and the application server, are in the control of the service providers. The goal of the Web Transactions Analysis and Optimization (TAO) project is *to efficiently manage the overall response times of Web transactions, by adjusting the resources under the control of the Web service provider.*

To effectively manage response times for Web transactions, we must address the following technical challenges:

1. *Instrumentation:* we require correlated measurements for transaction response times, residence times, and utilizations of various component resources, to understand performance behavior of the transaction flow.

2. *Monitoring Infrastructure:* a scalable monitoring infrastructure is required to collect data from the instrumentation with low overheads and minimum disruption to the Web transaction.

3. *Analysis:* The data collected from the instrumentation must be analyzed and abstracted for presentation to human decision makers (visually or analytically) and automated *classification* and *optimization* modules.

4. *Optimization Models:* several queueing centers multiplex their services among a variety of Web applications and transactions. Understanding the complex interplay of the queueing centers and their transactions on the overall response time requires mathematical models. We can subsequently use such models for automated management modules that can effectively tune the working of the controllable web service resources.

In this paper we describe the overall approach of the TAO project to address these challenges.

TAO uses WebMon, a novel *instrumentation and monitoring infrastructure* for Web transactions [8], that provides correlated, transaction performance information. WebMon monitors a Web transaction from both the client and server perspectives. Moreover, it correlates the client-side data with the server-side data to determine correlated performance measurements for each individual Web transaction. The instrumentation for monitoring individual components, or correlating the component measurements, does not require major modifications to the existing Web infrastructure or service. Small code fragments can be inserted automatically in appropriate components for instrumenting a web service. The monitoring infrastructure executes in parallel to the Web infrastructure. The monitoring traffic can be isolated (upto 96%) from the transactional traffic.

We use data mining, visual data mining, and statistical approaches for *analyzing* the information collected from WebMon. Our data mining approaches allow us to develop *clusters* or *segments* of user classes or transaction classes. This enables us to perform analysis and system optimization at the level of abstract transaction classes, rather than the multitude of individual Web transactions. For example, we can classify Web transactions into robot or human traffic; then, we can route transactions to appropriate back-end application servers based on this classification.

We use mathematical programming techniques to develop optimization algorithms for optimal allocation of software components on hardware elements. The optimization programs can make use of Web request classification to enable dynamic load balancing, load shedding and throttling, in addition to capacity planning. Hence, we provide integrated demand and capacity planning.

The rest of the paper presents an overview of the various components of TAO. We first describe the overall TAO approach in Section 2. Section 3 describes the analyses enabled by the TAO instrumentation. Section 4 describes the TAO approach for integrated demand and capacity planning. Section 5 overviews the TAO visual data mining of Web transactional data. In section 6 we review related work, and conclude with final remarks in section 7.

2 TAO Architecture

The back-end of modern Web applications have a multi-tiered architecture: the first back-end component is a load balancer. Behind the load balancer is a *web server farm*. The web servers can process a portion of the Web requests themselves, for example for static web pages or images. For dynamic web pages, a web server routes the request to one of several *application servers*. An application server generates a dynamic web page, sometimes collecting data from a database, other times collecting data from other web services. The generated web page is then routed through the web server back to the client component. Figure 1 illustrates the flow of the request through these various components.

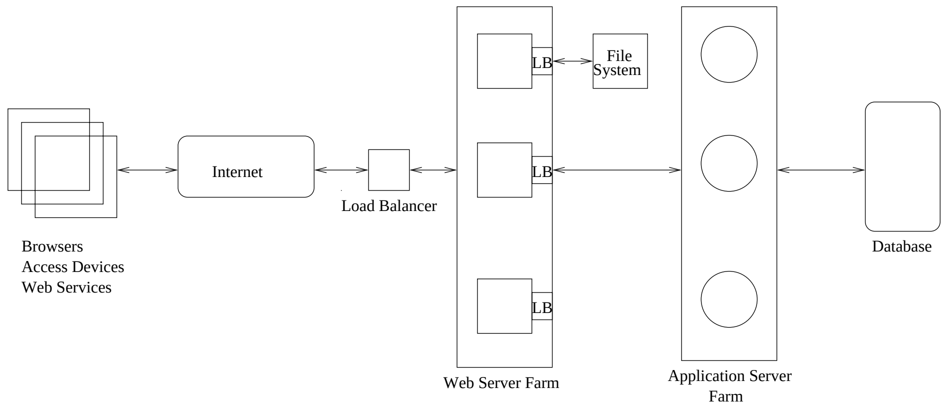

Figure 1: Typical transactional flow of a Web request.

Over the years, the web server architecture has stabilized into a multi-process or multi-threaded architecture, with usually an event-based concurrency control [18]. For the application server architecture, on the other hand, two competing architectures are emerging: (1) Microsoft's .NET architecture, and (2) Sun Microsystem's J2EE (Java 2 Enterprise Edition) architecture [13]. The two architectures are quite similar for our purposes: they both rely on distributed objects for back-end web content generation. In the rest of the paper, we will focus on the J2EE architecture, although the results and approach should be applicable to the .NET architecture also.

Figure 2 shows the main components of the J2EE architecture [13]. When a Web request is first routed to the J2EE engine, it comes in either as a JSP (Java Server Page) request, or a Java Servlet request. A multithreaded JSP/Servlet engine processes the request. The request is routed to a Java Virtual Machine (JVM) servlet *container* (`www.apache.org`). Each request is concurrently processed by a single thread. The JSP/Servlet engine may make requests to several *Enterprise Java Bean (EJB)* objects. Such EJBs may be co-located with the servlet engine or may be on a remote machine. In turn, the EJBs may invoke the services of a database engine, via a JDBC interface (`java.sun.com/products/jdbc`). The EJB container multiplexes multiple, concurrent requests for an EJB object using a thread pool.

For understanding the performance issues with typical J2EE architectures, we have developed an instrumented test-bed with: (1) an Apache Web server with the mod_jk load balancing module (`www.apache.org`), (2) three Tomcat (`www.apache.org`) and JBOSS (`www.jboss.org`) servers, and (3) one Postgresql server (`www.postgresql.org`). The test-bed has been completely instrumented with WebMon [8] instrumentation, to provide the following data:

1. End-user response time,
2. Web server identification and response time,

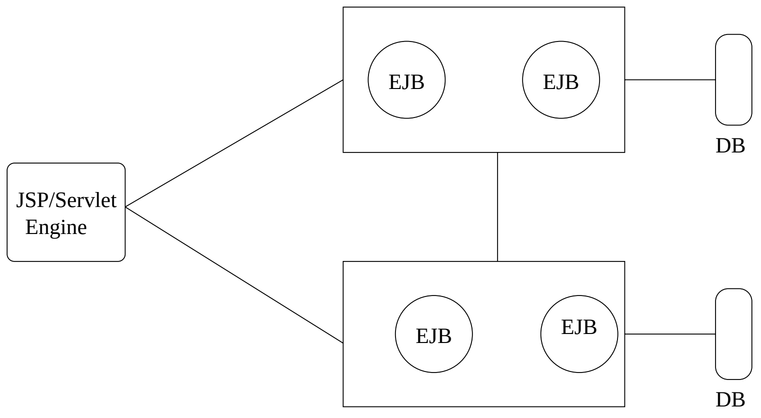

Figure 2: Typical J2EE application server architecture.

3. Application server identification and response time, and
4. JDBC time: time spent accessing the database through the Java Database Connectivity (JDBC) driver.

In the next few sections we'll describe how we use this information is useful for a variety of transactional analysis and optimization.

3 Analysis

We analyze WebMon data for:

1. Classification and segmentation of Web requests, and
2. Performance characterization of the J2EE architecture.

3.1 Web Request Classification

For performance characterization, we are interested in classifying Web requests along two dimensions: (1) transaction classification, and (2) user segmentation. In transaction classification, we map a given URL into a transaction class. For example, a URL `www.shop.com/estore/?item=1` may represent a "browse" transaction, while a URL `www.shop.com/estore/?product=2` may represent a "purchase" transaction. Multiple URLs can belong to a given class, while one URL belongs to only one class. In user segmentation, we are interested in identifying the segment of user population from which a given Web request originates. For example, in one segmentation we divide the clients as robots or humans. Another segmentation is based on purchasing behavior, i.e., humans who make frequent purchases versus those who do not. We are applying machine learning techniques for transaction classification [16]. For user segmentation, we are building on the statistical analysis work of Almeida et al [1]. For example, we studied the Web log of an online store covering a period of about one week. A random sample of about 80,000 sessions were provided to us, as well as a sample of about 200 sessions that had 100 transactions or more. The long sessions have a high probability of being robot-generated, since the sessions in the larger sample, which presumably is mostly human-generated, rarely exceed twenty transactions and on average lasts fewer than 10 transactions per session.

By studying the inter-arrival times in the long sessions, we were able to identify particular ones that looked like they originated from a shopping bot, as well as some which we believed to have originated from multiple human users connected through a proxy server. In some cases, these preliminary classifications were confirmed by the agent strings appearing in those sessions. We believe that using the profiles of inter-arrival times of sessions is one valuable method to identify robots relatively early. As we will see later, being able to predict the characteristics of an incoming session is important in the implementation of sophisticated load balancing policies.

Some authors (`http://robotics.stanford.edu/users/ronnyk/goodBadUglyKDDItrack.pdf`) have criticized the use of Web logs as the basis for analysis and segmentation, arguing that the data collected is typically inadequate and the transactions are not correlated with resource usage in the application and database servers. The WebMon instrumentation overcomes many of these deficiencies. With more widespread WebMon use, we will be able to study the benefit of having client side data, e.g., think times, in the classification of sessions.

3.2 Empirical study of various load balancing strategies

The web request classification derived from the previous sub-section can be utilized for effective load balancing at the back-end of the web services. For example, we can route human traffic in a different way than we route robot traffic. Similarly, we can treat traffic with resource intensive workload in a different way than light usage traffic. For load balancing, Web requests arrive at a multiple-server queue. Assuming a characterization of each Web request based on the classification described above, the problem is to assign the request to one of several back-end servers. The goal is the minimize some criterion like the mean overall response time of the request.

There are several well-known assignment policies. Our starting point is from reference [11], which compares the round-robin, random, size-based and least-work-remaining policies; they find empirically that the size-based policy, in which all requests within a given size range are assigned to a particular server, does well for a heavy-tailed request size distribution. The explanation for this is that the size-based policy minimizes the variance of the request sizes within the queue of each server while keeping the multiple servers equally utilized.

Our application differs from the traditional setup in several important ways. For example, Web requests are typically grouped into *sessions*, where each session is composed of a sequence of Web requests. The assignment for routing to a J2EE server is done once per session upon the arrival of the first Web request in that session. Web requests from different sessions may interleave. To implement the minimizing-variance idea of the size-based policy, we need to know (or estimate) upon the start of the session what the subsequent Web request sizes in that session look like, so that sessions comprising mostly small requests may be assigned to different servers from those comprising mostly large requests.

It may turn out that the advantages of a sophisticated assignment policy may be large enough that we would want to re-assign a session after we have observed it for a while and have formed a more accurate estimate of the subsequent request sizes. For example, if shortly into a session it is determined that the session is a robot that will issue one type of request a large number of times, it might be worthwhile to re-assign it to a server dedicated for those requests. Doing this would require a session-handover protocol that is absent in the Web application servers today.

We used our experimental test-bed to study the relative performance of the four load balancing policies. We selected three types of sessions with different lengths but approximately the same utilization. A type I session comprises various request types like "login", "search", "view product" and "add to cart"; a type II session comprises several "search" requests; a type III session several "view product" requests. Multiple client scripts were run in parallel for about one hour; each of them generating a stream of random sessions chosen independently from the three types.

Figure 3 shows bar charts of the mean client response times, one for each request type, for a lightly loaded system. Within a single chart, each bar represents one policy and is the (trimmed) mean of about 200 Web requests. We can see that the four policies are comparable, with the size-based method having a slight edge most of the time. We are currently investigating systems with higher utilization levels, greater variety of session types and incorporating uncertainty in the Web request sizes.

In the next section, we present an overview of how we plan to use the resource requirements derived from the test-bed measurements for optimal configuration of software components on hardware servers.

4 Integrated Demand and Capacity Planning

We now explore the problem of integrating user transaction demand with *online, realtime* resource capacity planning. This concept is known in the literature as "Capacity on Demand [14]." The integration of demand and capacity planning consists of optimizing (in real time or close to real time) the assignment of buffer (or shared) resources to satisfy the requirements of several applications. This investigation is motivated by the HP need of de-

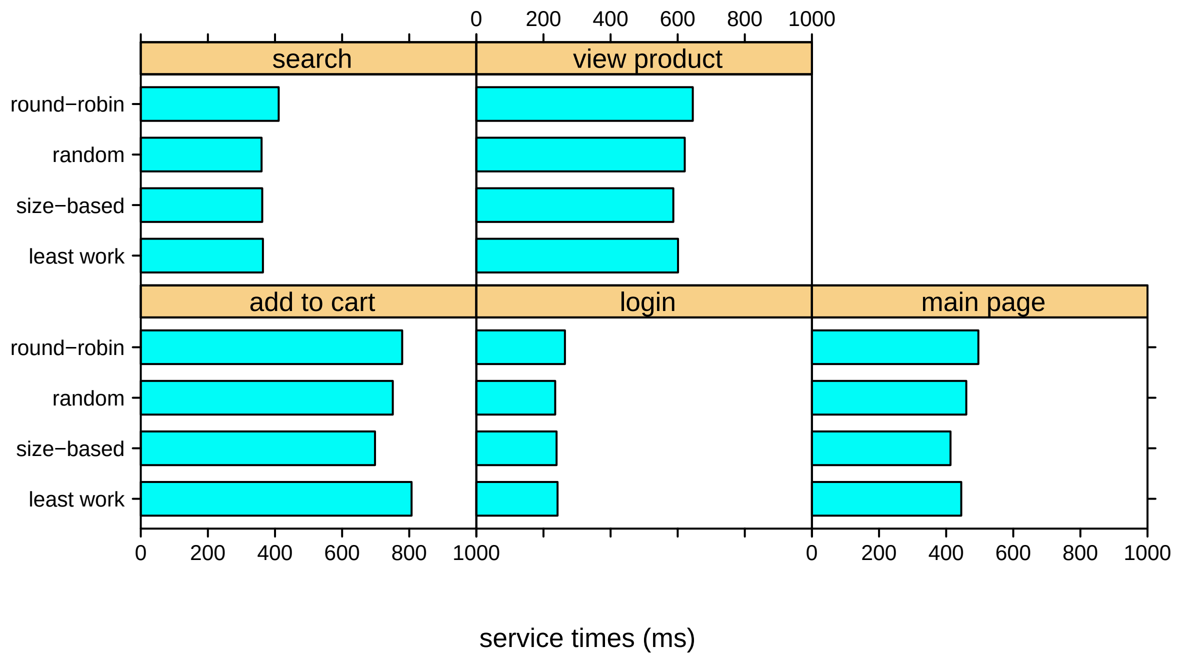

Figure 3: Trimmed mean response times for each request type.

veloping a highly available, and manageable infrastructure for e-business computing utility. At HP, we call these large data centers Utility Data Centers (UDCs). The main concept of an UDC is to enable multiple J2EE-like application server customers to be hosted on a collection of shared resources. The hosting environment can be divided into secure domains, each supporting one customer. These domains are dynamic: the resources assigned to them may be increased when workload increases and reduced when workloads decreases. This dynamic resource allocation enables flexible Service Level Agreements (SLAs) with customers in an environment where peak workloads are greater than the normal steady state.

Traditionally, user transaction analysis and control is not integrated with online capacity planning. The purpose of this research is to integrate instrumentation, transaction analysis and control with online capacity planning, in situations where there is buffer capacity and resources are shared among several applications–which is the case for UDCs.

Using Webmon instrumentation technology, we collect user and transactions data, and resources performance metrics for several applications running at a UDC. A Web transaction's back-end web servers, application servers, and database servers of each application are in control of the UDC.

As described in the previous sections, we use classification techniques to group related Web requests together into groups according to user segmentation, transaction usage, or resource requirement. Such classification is important for applications running at a UDC that have heavy traffic, since based on such classification different priorities can be assigned to each user segment, and user transaction throttling and load balancing can be based on user segment priority. This automated capability of user segmentation offers a mechanism to control the demand of resources of each application running at an UDC. In addition, the transaction classes characterize workload mix and volume for each application running at a UDC.

Assume that the layers of architectural components of each application running at an UDC is fixed. For example, a J2EE application typically has three layers: (1) Web, (2) JSP/Servlet and EJB engines, and (3) Database engine. For each of such layers, we assume a fixed type of servers and the desirable attributes of the server. Moreover, we assume that in any given layer, all software components will be mapped to the same server type. Then, for a given workload mix and volume for an application, we want to determine the *number of servers* required at each layer of the application architecture in such a way that the average response time is within the ranges specified by the SLAs, while the total number of servers assigned to the application architecture is minimized. We require a mathematical model that predicts the average response time for a given number of servers at each layer of the architecture, and an optimization algorithm that determines the minimum number of total servers required for the application average response time to be within the range of the SLAs. In section 4.1, we explain the main ideas of a proposed mathematical (queueing) model.

Given the servers requirements for each application architecture that satisfy the SLAs for the application workload mix and volume, we want to determine the specific servers available in the physical topology of servers and switches of an UDC in such a way the incoming and outgoing bandwidth capacity constraints of nodes in the physical topology, and the servers requirements of each application running at the UDC are satisfied, while communication delays between servers assigned to each application is minimized. We develop a large scale mixed mathematical programming model that can be solved using commercially available solvers in seconds. In section 4.2, we explain the main ideas of this mathematical programming model.

4.1 The Queuing Problem: Modeling Response Time

In this section, we propose a queueing model to approximate the average response time for a given number of servers at each layer of the architecture, and an optimization algorithm that determines the minimum number of total servers required for the application average response time to be within the range of the specified SLA.

The queuing problem is to find the probability distribution function of the response time to a request, given (1) the numbers of servers at the three back-end tiers (N_{web}, N_{app}, and N_{DB}), and (2) the mix and intensity of the incoming requests ($\lambda_1, \lambda_2, \ldots$). That is, we seek a function $f(.)$, so that $Pr(R \leq r) = f(r|N_{web}, N_{app}, N_{DB}, \lambda_1, \lambda_2, \ldots)$, where R is the random variable response time (or system residence time) of a request, r is any arbitrary level such as 0.1 second or 2 seconds.

If we "tag" a random request and follow it through the processing network, we will find that the routing and waiting pattern is highly complex at each tier of the system. For example, a request at the App server might trigger multiple query requests to the DB server; these queries might be generated depending on the results of a previous query. A request might also simultaneously demand multiple system resources at a server, and require multiple passes through the same server (one pass as a new request, second pass as a request returned from a downstream server).

Our goal is to have a simple but fairly accurate mathematical model of the total waiting time in the system. Hence, we model the system in aggregation, with simplified routing of requests through the system. In particular, we model the system as an open queuing network, with 3 tiers arranged in series, and parallel, identical servers inside each tier. With this open queuing network assumption, we aggregate the multiple-pass processing of returned requests into a one-pass simple flow (from a Web server to an App server to a DB server to exiting the system). Furthermore, we model each server as a processor-sharing queue with one critical resource (e.g., CPU or disk). The service demand of a request at a server is the sum of processing times of the multiple passes of this request at the server.

With this simple model, we can write the expected response time as the sum of response times at each of the three tiers:

$$E[R] = \frac{E[S_{web}]}{(1 - \frac{\lambda_{web} E[S_{web}]}{N_{web}})} + \frac{E[S_{app}]}{(1 - \frac{\lambda_{app} E[S_{app}]}{N_{app}})} + \frac{E[S_{DB}]}{(1 - \frac{\lambda_{DB} E[S_{DB}]}{N_{DB}})}$$

where λ_{web} is the arrival rate of new requests into the Web server tier, which is the sum of all customer request types that require processing at the Web server tier; $E[S_{web}]$ is the average service demand at the web server tier, averaged over all request types and including multiple passes of processing; and N_{web} is the number of servers at the web server tier. Similar notation applies for the App and DB server tiers.

The above formula assumes a processor-sharing queue at each server. We also assume that the servers within each tier are identical and share approximately the same workload; this implies that the arrival rate at each server is λ_{web}/N_{web}.

It is possible that only a fraction of all customer requests require routing into the App or DB server; in that case, we generally expect $\lambda_{DB} \leq \lambda_{app} \leq \lambda_{web}$. All three arrival rate parameters are determined from the input workload mix and intensity $(\lambda_1, \lambda_2, \ldots)$.

To obtain an estimation of the service demand at the web server tier $E[Sweb]$, we can use the relationship $u_{web} = \lambda_{web} E[S_{web}]/N_{web}$, where u_{web} is the average utilization rate of the critical resource (CPU) at the web server tier, to write $E[S_{web}] = u_{web} N_{web}/\lambda_{web}$. Note that this is a proposed model that still requires validation with experimental data.

4.2 Optimization

In the Capacity on Demand decision optimization module, these queuing results can be used as a service-level constraint, such as $E[R] \leq 0.5$ second. When a response time (T) is guaranteed at a critical level (α), say "response time not to exceed 2 seconds 95% of the time," then we write the optimization constraint as $Pr(R \leq 2) = 0.95$. To obtain the probability $Pr(R \leq 2)$, we use Markov's inequality on the non-negative random variable R:

$$Pr(R \leq T) = E[R]/T$$

Hence, the mathematical constraint $E[R]/T \leq \alpha$ will guarantee $Pr(R \leq T) \leq \alpha$. Our preliminary numerical results indicate that the bound E[R]/T on the probability $Pr(R \leq T)$ is usually loose for the type of queuing time distribution (of R) we are considering, hence replacing the constraint $Pr(R \leq T) \leq \alpha$ by the constraint $E[R]/T \leq \alpha$ will usually results in much better service level than that specified by α.

4.3 A Mixed Integer Programming Problem: Modeling Optimal Allocation of Resources at an UDC

In this section, we explain how to optimally allocate physical resources from a UDC, given the resource prediction and optimization needed to satisfy the resource requirements of a workload volume and mix and SLAs.

We consider the problem that given the physical topology of a UDC, a given application with a tiered architecture, and resource requirements for the application, how to allocate servers in the topology into the tier architecture in such a way that the application resource requirements are satisfied and network latency is minimized. This problem was originally formulated by Zhu and Singhal [19]).

For example, consider an instance of this problem depicted by Figure 4:

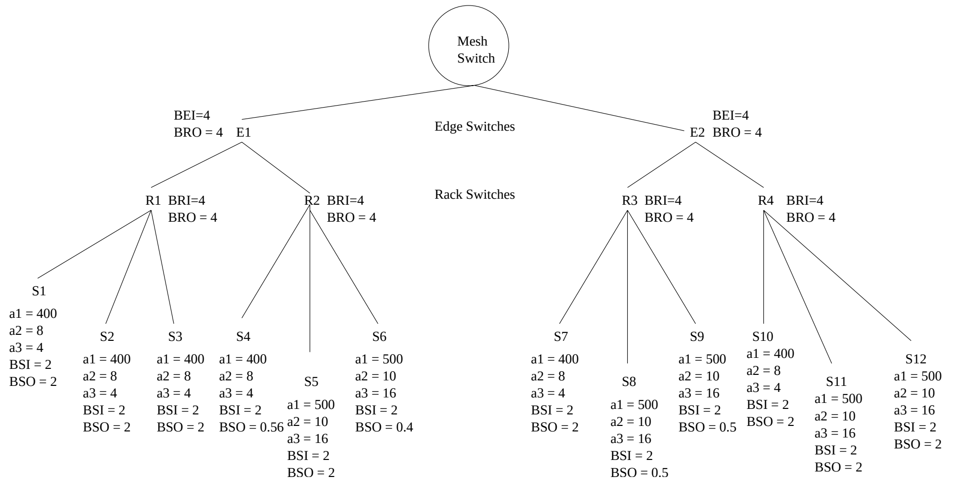

Figure 4: Physical topology of the example problem.

- The layout of the physical topology is a hierarchical tree; the root of the tree is a mesh switch.
- The number of edge switches is 2. We label them as e_1 and e_2.
- The number of rack switches is 4. We label them as $r_1, r_2, \ldots, r_4$.
- The number of servers is 12. We label them $s_1, s_2, \ldots, s_{12}$.
- The number of attributes per server is 3. These attributes may be CPU speed, memory size, disc capacity, etc. We label the attributes as a_1, a_2, and a_3.

- Each server, rack switch, and edge switch has an incoming and outgoing bandwidth capacity. We label these bandwidth capacities as B_{si} and B_{so}, B_{ri} and B_{ro}, B_{ei} and B_{eo}.

The parameters for the application tier architecture are presented in Figure 5

- The number of tiers in the architecture is 3. We label them as l_1, l_2, and l_3.
- The number of servers required at each tier i are: $N_3 = 1, N_2 = 2, N_1 = 2$.
- Min/Max server required attribute are labeled as L and U respectively.
- The matrix T represents the amount of traffic going from each server in a tier to each server on a consecutive tier.

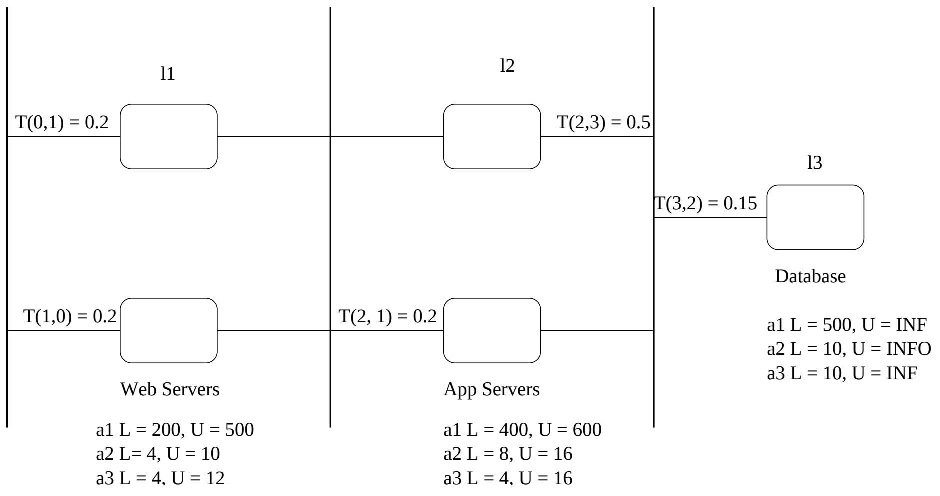

Figure 5: Tiered Architecture.

Our problem is to identify which servers in the physical topology we should allocate to the application architecture in such a way that latency communication delay between servers is minimized, while bandwidth capacity constraints and Min/Max server attribute requirements are satisfied.

The main assumptions of our model are as follows:

1. The physical topology is a tree
2. Consider one application at a time

3. The application has a tiered architecture
4. Servers at the same tier have same functionality, consequently they have the same attribute requirements
5. The amount of traffic generated by different servers in the same tier is similar. The amount of traffic coming into each tier is evenly distributed among all the servers in the tier
6. No traffic goes between servers in the same tier. Servers communicate only with servers in adjacent tiers.

Our approach has three steps, each represents a mathematical optimization model.

1. We use a mixed integer programming model to generate an approximate good initial solution. This initial solution might not satisfy all bandwidth capacity constraints.
2. We use a nonlinear programming model to find a local optima and feasible approximate solution. In this case, the solution satisfies all constraints.
3. We use another mixed integer programming model to generate an exact solution from the approximate solution found by the nonlinear programming model.

With this approach, we can solve in seconds problems with thousands of servers using commercially available solvers for mathematical optimization. More details for this solution are available in [15].

5 Visual Data Mining

Due to the high volume of Web transactions, typically the performance data collected for a Web service, even for a short time, results in large amounts of performance data. Finding the valuable information hidden in the performance data, however, is a difficult task. Visual data mining techniques are indispensable to solving this problem. In most data mining systems, however, only simple graphics, such as bar charts, pie charts, scatter plots, etc., are used to support the data mining process. While simple graphics are intuitive and easy-to-use, they either show highly aggregated data or have a high degree of overlap that may occlude a significant portion of the data values (as in the case of scatter plots). Important information often gets lost. With large volumes of web transaction, bar charts and scatter plots quickly become cluttered and difficult to visualize. Hence, we require new requirements for visual data mining of performance data:

1. Place web data items with similar behavior and relationships close together,

2. Unclutter the display with no overlapping,

3. Interact with the user for analyzing different scenarios, and

4. Scale to a large number of web transactions and clients.

We describe new visual data mining techniques that address the above requirements. The first method is "**Pixel Bar Chart**" [10]. Pixel bar chart is used to visually mine multi-attributes transaction data sets, such as response time, and page size. The second method is a "**Physics-Based Mass-Spring Visual Clustering**" technique [7, 17]. It displays web data items, such as clients and URLs, with close relationships together onto a 3D graph. In the meantime, a set of content sensitive interactive visual techniques–linked multiple views, layered drill-down, are provided to allow web analysts to interact with their data. We have integrated both techniques into a VisMine [9] platform to visually analyze millions of Web transactions.

5.1 Using Pixel Bar Chart To Visual Mining Web Transactions

Pixel Bar charts are a generalization of traditional bar charts and x-y-plot, to allow visualization of large amounts of transaction data. The basic idea is to use the pixels within the bars to present detailed data record information. Pixel Bar Charts retain the intuitiveness of traditional bar charts while allowing very large data sets to be visualized in an effective way. The detailed information on each data item can be displayed by drilling down as needed.

5.1.1 Visual Mining Transaction Distribution

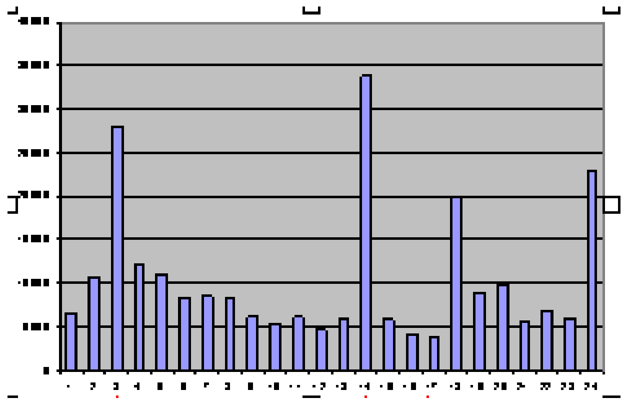

Figure 6: Bar chart of average hourly response times.

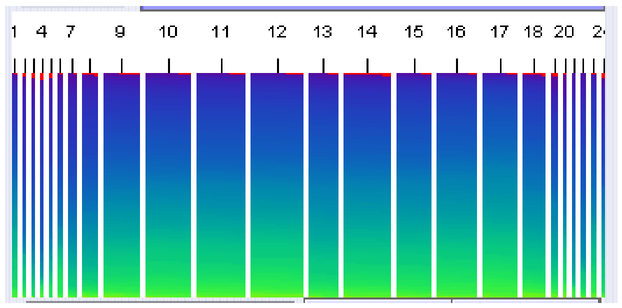

Figure 7: Pixel Bar chart hourly Web transactions (response times).

Figure 7 shows a sample pixel bar chart of 32,846 web service transactions. In Figure 6, the bar chart shows only the average response time per hour. No other related information is shown. In Figure 7, the pixel bar chart shows the web service response time activities in one day (24 hours). Each pixel represents a transaction record. The x-axis (dividing attribute) is time slot (hour); the y-axis (ordering attribute) is the web service response time ordered from bottom to top. The colors (from green to dark burgundy) in the different bar charts represent different response time values in milliseconds. The red area represents transactions which response time exceeds 100 seconds. From Figure 7 we can obtain the following information about the web service:

- The busiest times is during the middle of the day during the hours 11, 12, 14... (with wider width bars),
- Amount of transactions exceed the threshold (100 sec, colored red),
- Amounts of transactions achieving good response times (colored green), and
- Web services have bad response time at all hours, i.e. all have red areas.

By clicking on a specific pixel, the user can get detail information on the transaction represented by that pixel, such as at the 14th hour. For example, response is 8 ms, client IP is 15.8.154.86 and URL is troi.rose.hp.com.

It is further interesting that bar chart indicates the 14th hour has the worst response time. Actually, there are large area colored with green and blue in Figure 7, which means that there are many transactions with good response time. Only a very small number of transactions colored red with very long response time (exceed 100 sec). This explains the high average response time for the 14th hour.

5.1.2 Visual Mining Web Transaction Multi-Attribute Correlations

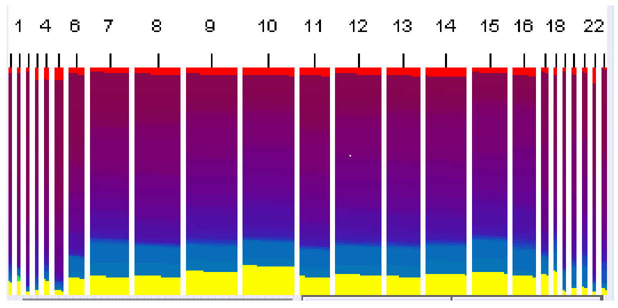

Figure 8: Pixel bar charts with page sizes.

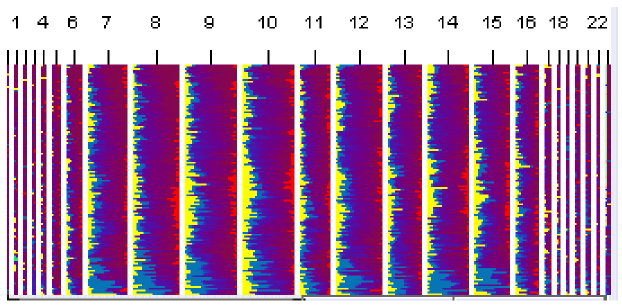

Figure 9: Pixel Bar with page sizes and response times).

In Figure 8, the pixel bar chart shows the web service page size activities in 24 hours. Each pixel represents a web transaction record. The x-axis is the time slot (hour); the y-axis is the web page size ordered from bottom to top. The colors in the different bar charts represent different page size values. From Figure 8 we can obtain the following information regarding page sizes of the monitored Web transactions:

- Several page sizes exceed the threshold of 100,000 bytes, as indicated by red color,
- This Web service has different page sizes during the day (7th hour to 16th hour), and
- The Web service has large page sizes during the nights and early morning (1st–5th hour, 17th to 24th hour).

Figure 9 illustrates the correlations between the response time (Figure 7) and the page size. The x-axis is time slot (hour); the y-axis is web response time ordered from bottom to top. The colors in the different bar charts represent different page size values. Red area presents the transactions whose page size exceeds 100,000. From Figure 9 we can conclude that there is no close correlation between response time and page size. We have discovered that to different page sizes displayed in all ranges of the response time (yellow, green, blue and burgundy colors across all different levels of response time).

5.2 Using Visual Clustering To Discover Web Access Patterns

To analyze a large highly related web client space, we are experimenting with a new Web Access Visualization technique. This method arranges the data items (clients) extracted from the web transaction data onto a spherical surface. Items are represented as vertices. A transaction is a web log record that contains a client, a URL and network access response time. The transaction data is described as follows:

Transactions $T_1, T_2, \ldots, T_n$, n = no. of transactions

Items $I_1, I_2, \ldots, I_m$, m = no. of data items (e.g., clients)

For each item I_i, a transaction set S_{Ii}, defined as

$$S_{Ii} = T_{i1}, \ldots, T_{ij}, i = 1 \ldots m, j = 1 \ldots k, k \leq n$$

This technique calculates the correlation between pairs of data items, based on some aggregate transaction metric, e.g., median response time (M). The data items will be laid out according to their similarity for the chosen transaction metric. The relaxation algorithm calculates the final layout.

$$M(S_{Ii}) = \text{median response time for transactions in } S_{Ii}$$

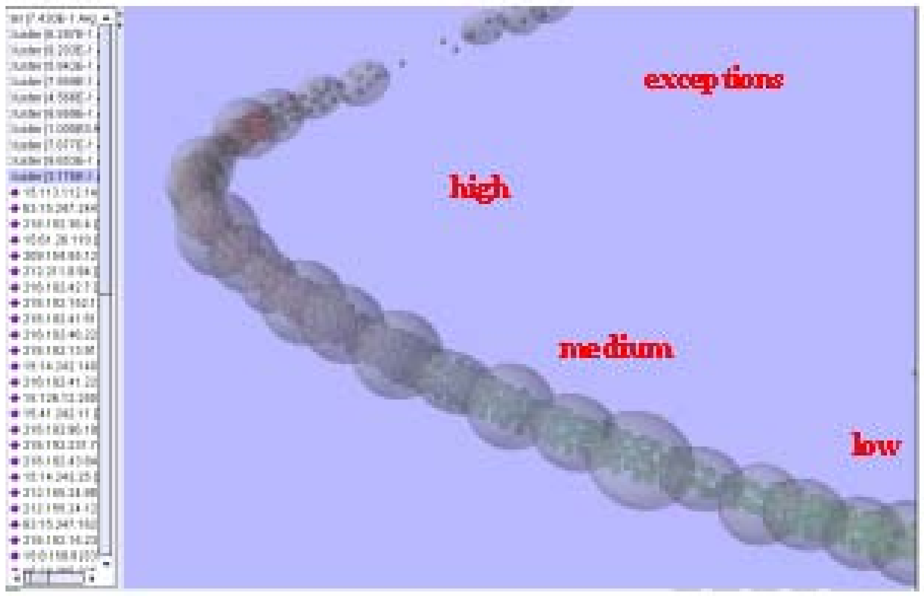

Figure 10: A visual web client clustering layout, according to response times received by client nodes.

$$Similarity(I_i, I_j) = min(1.0, \frac{(|M(I_i) - M(I_j))}{max(M(I_i))}), i = 1 \ldots m$$

Figure 10 illustrates a 3D graph generated from a web transaction observation data set. It contains 35,000 transaction records. There are 986 clients that make transactions on the web. VisMine clusters clients into 20 clusters with a similar response time (from low to high). The "distance" between each pair of clients represents the data access relationship. The most tightly related client is the client with the highest correlation with other clients. These clients usually have similar response times.

6 Related Work

The Web has become a pervasive infrastructure for information-related transactions. As such, many solutions for performance management of Web transactions are emerging. Early work on Web transactions performance management addressed two areas: (1) scalability of the web server [3, 6, 5], and (2) Content Delivery Networks (e.g., Akamai `www.akamai.com`). The web server scalability has resulted in architectures that can routinely support tens of thousands of connections per second. Content delivery networks speed up the delivery of content by placing content as close to the edge (e.g., the user's browser) as possible.

The Web server scalability provided by these solutions does not take into account the Web request classification as advocated by the TAO project. They typically use the URL as the only means of classifying and routing requests. We use additional information, such as response time (at the client and server sides), inter-arrival times, failure times, and so forth, to classify and route requests. Such classifications become important with J2EE-like architectures, as the number of URLs are increasing with more and more information encoded in "fat URLs."

Efficient, scalable back-end architectures is one aspect of TAO. Another important aspect is to provide robust, complete, correlated instrumentation for monitoring a J2EE-like environment. Since the architecture is relatively new, only now we are seeing monitoring architectures evolving in this direction. For example, Precise Solutions of Israel (`www.precise.com`) provides a complete J2EE instrumentation similar to WebMon. The visual, statistical, and data mining analysis provided by TAO is distinct from the kind of analysis performed by Precise Solutions.

Finally, TAO addresses the problem of efficiently mapping software components onto hardware resources in real-time, i.e., capacity on demand. The Oceano project [2] also addresses this problem. However, the Oceano project does not use analytical technique such as queueing theory and mathematical optimization as we have done in this research. We believe that this analytical techniques are critical for the implementation of "Capacity on Demand" at large data centers such as UDCs.

7 Conclusions

In this paper we have described an overview of the Web Transactions Analysis and Optimization project (TAO). TAO monitors a Web transaction from several performance perspectives, and makes the monitored information available to a variety of analysis modules. Such analysis modules make it easier for human decision makers to understand the performance characterization of their web service, e.g., through pixel-bar charts or visual clustering approaches. In addition, the analysis can be made available to automated modules for efficient load balancing and managing capacity on demand. In particular, we use segmentation of Web requests from a users, behavioral and resource consumption perspective to efficiently manage the performance of Web transactions.

Acknowledgements

We thank the numerous people in the Software Technology Lab and OpenView Business Unit of HP for their help with this project.

8 References

[1] V. Almeida, D. Menascé, R. Riedi, F. Peligrinelli, R. Fonseca, and W. Meira. Analyzing Robot Behavior in e-Business Sites. In *Proceedings of SIGMETRICS Joint international conference on on Measurement and modeling of computer systems*, pages 338–339, 2001.

[2] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, and B. Rockwerger. Oceano: SLA Based Management of Computing Utility. IBM technical report.

[3] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable Content-Aware Request Distribution in Cluster-Based Network Servers. In *Proceedings of the 2000 USENIX Annual Technical Conference*, San Diego, Calif., USA, June 2000.

[4] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.

[5] L. Cherkasova and M. Karlsson. Scalable Web Server Cluster Design with Workload-Aware Request Distribution Strategy WARD. In *Proceedings of the IEEE International Workshop on Advanced Issues in E-Commerce and Web-Based Information Systems*, San Jose, Calif., USA, June 2001.

[6] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A Scalable and Highly Available Web Server. IBM Research Division, Yorktown Heights, NY 10598.

[7] M. H. Gross, T. C. Sprenger, and J. Finger. Visualizing Information on a Sphere. In *IEEE/VisInfo97*, Arizona, 1997.

[8] T. Gschwind, K. Eshghi, P. Garg, and K. Wurster. Web Transaction Monitoring. Technical Report HPL-2001-62, Hewlett-Packard Laboratories, Palo Alto, Ca, April 2001.

[9] M. Hao, U. Dayal, and M. Hsu. A Java-based Visual Mining Infrastructure and Applications. In *IEEE/VisInfo99*, 1999.

[10] M. C. Hao, J. Ladisch, U. Dayal, M. Hsu, and A. Krug. Visual Mining of E-Customer Behavior Using Pixel Bar Charts. In *ACM KDD/2001*, August 2001.

[11] M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On Choosing a Task Assignment Policy for a Distributed Server System. *Journal of Parallel and Distributed Computing*, 59(2):204–228, 1999.

[12] P. Mills and C. Loosley. A Performance Analysis of 40 e-Business Web Sites. *CMG Journal of Computer Resource Management*, 102, Spring 2001. Also available from `www.keynote.com`.

[13] R. Monson-Haefel. *Enterprise Java Beans.* O'Reilly & Associates, Sebastopol, Calif., USA, 2000.

[14] J. Rolia, S. Singhal, and R. Friedrich. Adapative Internet Data Centers. In *Proceedings of SSGRR 2000, Computer and eBusiness Conference*, L'Aquila, Italy, July-Aug 2000.

[15] C. A. Santos, X. Zhu, and H. Crowder. A Mathematical Optimization Approach for Resource Allocation at Large Scale Data Centers. In preparaton.

[16] A. Seetharaman and P. Garg. Semi-Automatic URL Classification. In Preparation.

[17] T. C. Sprenger, R. Brunella, and M. H. Gross. H-BLOB: A Hierarchical Visual Clustering Method Using Implicit Surfaces. In *IEEE/VIS2000*, 2000.

[18] M. Welsh, D. Culler, and E. Brewer. Seda: An Architecture for Well-Conditioned, Scalable Internet Services. In *Proceedings of the Eighteenth Symposium on Operating Systems Principles (SOSP-18)*, Banff, Canada, October 2001.

[19] X. Zhu and S. Singhal. Optimal Resource Assignment in Internet Data Centers. Technical Report HPL-2001-59, HP Labs, Palo Alto, Calif., USA, March 2001.