Journal of Web Engineering, Vol. 7, No. 2 (2008) 093–132
© Rinton Press

REACTIVITY-BASED APPROACHES TO IMPROVE WEB SYSTEMS' QUALITY OF SERVICE

ADRIANO CÉSAR MACHADO PEREIRA LEONARDO DE ARAÚJO SILVA
WAGNER MEIRA JR. WALTER DOS SANTOS FILHO
Department of Computer Science, Federal University of Minas Gerais
Av. Antônio Carlos 6627 - ICEx - CEP 31270-010
Belo Horizonte, Minas Gerais, Brazil
{*adrianoc, leosilva, meira, walter*}*@dcc.ufmg.br*

Received November 29, 2006
Revised May 14, 2008

Understanding the characteristics of Internet services workloads is a crucial step to improve the Quality of Service (QoS) offered to Web users. Moreover, studying and modeling the user behavior is important to analyze the performance and the scalability of web servers. This knowledge may be used, for instance, to build workload generators that help evaluating the performance of those servers. Current workload generators are typically memory-less, being unable to mimic actual user interaction with the system. As the basis of this work, we propose a hierarchical characterization and simulation model focused on the user behavior, named *USAR*.

In fact, there is strong evidence that a significant part of the user behavior depends on its satisfaction. Users reactions may affect the load of a server, establishing successive interactions where the user behavior affects the system behavior and vice-versa. It is important to understand this interactive process to design systems more suited to user requirements. In fact, the user reactivity, that is, how the users react to variable server response time, is usually neglected during performance evaluation. In this work we study and explain how this reactive interaction is performed by users and how it affects the system's performance.

Web applications demand requirements, such as performance and scalability, in order to guarantee QoS to users. Due to these requirements, QoS has become a special topic of interest and many mechanisms to provide it have been proposed. In this work, we address the use of reactivity to improve Internet services. We propose and evaluate new admission control and scheduling mechanisms. We designed and implemented the *USAR-QoS* simulator that allows the evaluation of the new strategies considering the dynamic interaction between client and server sides in Internet services. We simulate the new strategies using a TPC-W-based workload. The experiments show the benefits of the reactive policies which can result in better QoS for Internet Services, improving the user satisfaction. We also propose a hybrid admission control and scheduling mechanism that combines both reactive approaches. The results show benefits in terms of response time and user satisfaction.

Keywords: Web systems, QoS, characterization, performance, user behavior, workload generation, reactivity

Communicated by: D. Schwabe & J. Freire

1 Introduction

Understanding the nature and characteristics of Internet services workloads is very important to design systems with better performance and scalability, which are features that affect directly the Quality of Service (QoS) experienced by the users. Workload characterization techniques, such as [34], are popular strategies for this task.

Workload characterization may also be the starting point to build synthetic workload generators. These generators are an effective way for exercising the system's capabilities through the request of operations that are similar to actual workloads. Nevertheless, most of the Web-related workload generators do not take into consideration systems variations in terms of QoS to reproduce the users requests, generating the same workload despite the server response time.

Accounting for user behavior is expected to aggregate valuable information to the modeling and generation of workloads through the analysis of criteria such as navigational patterns, actions requested, and the inter-arrival time (IAT) between requests. From the server or proxy perspective, the user behavior may be observed through the sequence of actions (i.e., clicks of a link or requests for web pages) performed by a user. Although desirable, it is not usually possible to observe other criteria such as the user activities between his actions. Characterizing and replicating the user behavior-related criteria is a challenge that we address in this work. In particular, we focus on the relation between IAT and the server latency (response time), i.e. the time to process and answer a request, for each action.

In the first part of this paper we describe *USAR*, a hierarchical workload characterization model. The model also comprises a validation methodology through simulation of user reactions. As mentioned, we consider behavior as how the interaction of users with web applications is affected by variable latencies. In order to demonstrate and validate the *USAR* model, we present its application and validation using actual data from a proxy-cache server.

Workload generators so far have been based on several aspects of Web workloads such as arrival process and resource popularity. However, there is one aspect that has not been considered: how users react to the performance of the system, that is, how the behavior of the user changes as a function of the response of a server. Previous work [37] presents a characterization methodology for reactive workloads and demonstrates that users do behave reactively. Nevertheless, it is not clear how the changes in user behavior affect the workload submitted to the server, and how the process iterates, when the server, answering to a reactive behavior, changes again the user behavior and so on. Considering these aspects, reactivity in Web server performance analysis raises new challenges, in terms of both realistic workload generation and scalability mechanisms.

The second part of this paper addresses the issue of generating reactive workloads and evaluating its impact on the server performance. To achieve this goal, we extend *httperf* [35], a well-known workload generator, so that it simulates the reactions of users according to the response time to their requests. The model used to replicate user reactions is based on [37]. It makes possible to reproduce user behavior patterns that represent how a user reacts to the QoS perceived. Our improved version of *httperf* is used to generate workloads based on the TPC-W benchmark [22]. These workloads are applied to a server in order to assess the impact on its performance.

There is strong evidence that a significant part of the user behavior is reactive, that is, the user reacts to the instantaneous conditions at the action time. As a consequence, user behavior varies according to some factors related to the server and the application provided. In this context, one important aspect to evaluate is how users react to the performance of the system, that is, how the behavior of the user changes as a function of the response of a server.

In the third part of this work we study and explain how this reactive interaction affects the system's performance. Moreover, we evaluate how different user profiles affects the system's load, changing the performance and defining different characteristics to this reactive environment.

Once we have studied and understood the impact of reactivity in the system's performance, we investigate how the reaction of users to QoS measures such as response time may be exploited for sake of designing novel and more effective QoS strategies. Several schemes to deliver desirable QoS have been proposed but they fail to capture the actual dynamics between users and the service. Mechanisms such as admission control and scheduling, commonly used in Internet servers, do not care about the different characteristics of users. For example, scheduling policies generally adopt a simple First-In First-Out (FIFO) approach, without considering that users have different levels of tolerance to the Internet QoS [10]. First, we propose and analyze the use of admission control techniques based on user reactivity to guarantee QoS in Internet services for workloads with different characteristics. We present the *USAR-QoS* simulator that allows the evaluation of the proposed QoS strategies considering the dynamic interaction between client and server sides in an Internet service scenario. The experiments show the gains obtained by the reactive strategies, demonstrating the benefits and drawbacks of the reactive admission control approach.

The new admission control approach based on the reactivity is effective for controlling the response time, but the rejection of requests causes an increase in the impatience of users. In order to reduce the impatience of users, we propose new scheduling techniques based on user reactivity. We present two different approaches that prioritize users with different profiles of reactive behavior: the Patient-First Impatient-Next (PFIN) and the Impatient-First Patient-Next (IFPN), both with two different configurations. We evaluate them using the *USAR-QoS* simulator that allows the simulation of the proposed QoS strategies considering the dynamic interaction between client and server sides in an Internet service scenario. For sake of reproducing representative workloads, our experiments use the TPC-W-based reference benchmark. The results show the benefits obtained by the reactive scheduling approach that is effective to reduce de bursts expired rates due to the impatient behavior.

Finally, we also propose a hybrid admission control and scheduling mechanism that combines both reactive approaches, optimizing the advantages of each one. The mechanism based only on reactive scheduling achieves the best burst lost rate, since its mechanism gives priority to requests classified with impatient classes. The mechanisms that adopt the admission control are effective to reduce the response time, but may cause the increase in the burst lost rate, due to the increase in the amount of users rejecting bursts or sessions. The hybrid mechanism presents an equilibrium, reducing both the response time values and the burst lost rate.

The contributions of this work are the formalization of reactivity concept, the specification of a multi-level reactivity model, elaboration of characterization methodologies for modeling user reactivity, and the validation of the model and methodology applying them to relevant scenarios, such as Web services.

The paper is organized as follows. In section 2, we present an overview of the related work. Section 3 presents *USAR* and defines the model to characterize and generate the user behavior simulation. Moreover, we discuss how to use the model and its validation, presenting a case study. Section 4 assess the impacts of the reactivity in a experiment using a reactive version of *httperf* workload generator. Section 5 presents our experimental study, that complements the evaluation of the reactivity impact. Section 6 provides some QoS background and describes our new admission control and scheduling approaches. It also explains the experimental simulation, presenting the simulator implemented, the methodology of analysis, and the case study. Finally we present our conclusions in Section 7.

2 Related Work

This section describes related works to the main subjects of our paper. First, we present some workload characterization. Next, we focus on the user behavior and the influences of the service performance on it. Then, we discuss the problem of generating realistic workloads. Finally, we describe works related to aspects of performance and Quality of Service, including analysis and characterization studies.

2.1 Workload Characterization

The characterization and generation of workloads are essential to the evaluation of Internet systems, motivating several studies over the last few years. These works[6, 18, 32] analyze some of the characteristics of workloads of web servers, and Jussara Almeida et. al and Eveline Veloso et al.[5, 41] analyze streaming media workloads. Daniel Menasce et al. and [34] propose hierarchical methodologies to characterize workloads based on a multi-level strategy: request, function, and session. Nevertheless none of these works models the Web systems characteristics related to the interaction of the users with them. Additional related work to workload characterization is described on the next subsection with focus on the user behavior analysis.

2.2 User Behavior

The user behavior can be analyzed using several variables observed in a Web server log. One may analyze the list of requests submitted, navigational patterns, types of services (functions [31]) accessed, think-times, among other information. However, in these cases, aspects related to the Quality of Service provided are not considered.

Henderson [24] uses the latency to study the user behavior in the specific context of a game application, detecting that network delay has effects on players' behavior, causing users to quit the service. Cristiano Costa et al. [17] study the correlation between requests in streaming media applications, trying to determine trends in the user interaction process. This work [14] tried to model the click-stream in the context of Web advertising. Anand Balachandran et al. [8] characterize the user behavior in a public wireless network, considering distribution of the users, session duration, data rates, application popularity and mobility. The author intends to optimize the quality of access provided by wireless services. Helmut Hlavacs et al. [26] proposed a user behavior model framework, built in a top-down manner, consisting of various layers and based on mathematical models. This work is used to produce a user oriented workload generator [25].

However, these studies fail to model the behavior of users to the performance provided by the service. They do not capture aspects related to the reactivity of users to the quality of service. Despite the importance of this subject, it has not been completely studied, demanding works to understand and use it to develop techniques to deploy services with better quality of service.

2.3 Web Workload Generation

Workload generators are tools designed to generate synthetic logs composed of requests that simulate real user requests. Workload generation is fundamental to evaluate Web systems since they allow us to experiment different scenarios of load. SPECweb99 [1], WebBench [3] and TPC-W [22] are benchmarks for evaluating the performance of Web Servers. They provide representative benchmark for measuring a system's ability to act as a Web server. SURGE [9] and *httperf* [35] are workload generators, developed to exercise Web servers through the submission of a set of requests with different characteristics of load.

These workload generators are powerful tools, but they are not capable of simulating user behavior patterns related to the reactions of users according to the performance provided by the service. They adopt an arrival process independent of the performance perceived, generating the same workload, despite the variations observed in the Quality of Service. Their request generation strategy considers neither the performance of the server nor the user reaction to it, that is, the workload generated has always the same arrival process, regardless on how the server answers the requests. Therefore, the study of techniques to fill this gap is important to improve existing workload generators and to provide the possibility to conduct more precise evaluation of Web servers.

2.4 Performance Evaluation

Performance evaluation is a very important subject in order to verify the quality of service provided by a Web site when different workloads are applied to it. Many works address this problem, studying many related aspects.

Some works evaluate the response time impact on users behavior, such as [40].David Olshefski et al. [36] present a mechanism for Web servers to measure the response time perceived by the clients, using a model of TCP that analyzes the broken connections. Other works such as [29] analyzes the connectivity of the clients in order to take actions to improve the response time provided. Dror Feitelson et al. [19] shows that performance of a computer system depends on the characteristics of the workload it must serve. Therefore performance evaluations require the use of representative workloads in order to produce dependable results. Martin Arlitt et al. [7] present a workload characterization study of six different data sets with different characteristics, discussing performance issues and suggesting enhancements for Web servers, focusing on caching strategies.

Our study uses the response time to analyze the impact of the server performance on the reactions of users, correlating it to the inter-arrival time between requests. This is the basis of the model presented in Section 3, that is implemented on a workload generator and used in experiments to analyze the impacts of the new approach on the performance of a Web server.

2.5 Quality of Service

Improving QoS has been a challenging task addressed by many works in diverse research areas. QoS for Internet servers has evolved significantly and most of the works in this area proposes architectures and strategies, such as admission control, load balancing, resource allocation, and scheduling. Ludmila Cherkasova et al. [16] develop a session-based admission control that rejects new customers during overload periods. Tarek Abdelzaher et al. [4] proposes a mechanism to adapt the content provided instead of rejecting requests to decrease the server load. Nina Bhatti et al. [11] discuss classifying web requests into three levels of priority using information as IP address and requested web sites.

Integrated Services (IntServ) [13] put applications into two classes based on timeliness requirements. IntServ aims to provide per flow QoS by reserving bandwidth along a source-destination path to assure timeliness of data delivery. Differentiated Services (DiffServ) [12] mark data on the edge of a network with two classes of priority, which have different priorities of being serviced. For IntServ and DiffServ frameworks, many scheduling and admission control mechanisms have been proposed. Tiziana Ferrari [20] investigates the effect of aggregation on the performance using Priority Queuing (PQ) and Weighted Fair Queuing (WFQ) scheduling algorithms. Huamin Chen et al. [15] exploit the dependence among session-based requests and propose a scheduling algorithm to control overload in web servers. Nong Ye et al. [43] use only one queue for all web requests with an advanced scheduling rule to differentiate services.

Nevertheless, these solutions fail to provide a robust solution for the Internet QoS, since they do not consider user reactivity. We believe that the use of user-reaction-based information is an effective way to design QoS strategies more suitable to user demands, improving Internet QoS.

3 The USAR Characterization Model

In this section we present the *USAR* characterization model and its validation strategy.

3.1 Workload Characterization Model

As described in Section 2, previous efforts employed hierarchical methodologies to characterize workloads, considering user-side and server-side metrics, but ignoring the correlation between them. In order to fill this gap we propose a new characterization model, named *USAR* which is capable of identifying and modeling the reactivity that represents the way a user behaves according to the Quality of Service provided. The model provides a characterization methodology based on a hierarchical and multiple time scale approach to characterize Web services workloads [32, 34]. We divide the methodology into four levels: User, Section, Action, and Request. Within each layer, an analysis across several time scales and criteria has to be conducted. Each level is described as follows:

User (U): This level models the user behavior considering the offered Quality of Service, abstracting from the operations performed by him/her.

Session (S): This level models the user interaction that occurs within a time interval, that is, the actions of the user that are apart less than a pre-defined threshold τ composes the session. The characterization considers the following criteria: session length, session duration, session composition in terms of actions, bytes transferred per session, and IAT between sessions.

Action (A): This level models the actions performed by the user which are usually clicks that activate a link or requests for a web page. Besides characterizing the probability distribution of the action types, we also characterize the correlation between response time and IAT.

Request (R): This level models the objects associated with a user action. We study the request arrival process and the probability distribution of its features.

The idea of the hierarchy of levels (see Figure 1) is to guide the analysis of the workload into different perspectives. This eases the characterization process, once it may be done according to different views associated to each level. Therefore this process becomes clear and produces a more detailed characterization.

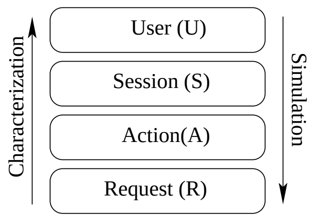


Fig. 1. The USAR Hierarchy of Characterization Levels

In the context of this work our main objective is to describe the characterization at the *user level*. As mentioned, this level intends to model the user behavior, more specifically the reactions of the user to variable latencies, that is, how the Quality of Service affects the user actions. Next we describe the 8-step methodology we propose to analyze and characterize this level.

1. Prepare log: Generate a temporary log Lu by putting together the sessions of each unique user;

2. Analyze users from the following perspectives: IATs between requests of the same user, latency associated with requests of the same user, IAT and latency ratio, and IAT and latency difference;

3. Discretize IAT and latency measures using a function that correlates them. In our case, the function takes into consideration the ratio and the difference between them, clustering user actions that are similar in terms of the two measures;

4. Transform user sessions into sequences of user action classes using the aforementioned discretization criterion;

5. Evaluate the sequences in order to group them according to a similarity criteria;

6. Process the log Lu applying a function $f(Lu)$, which maps sequence of classes to the groups defined in the last step;

7. Apply a clustering technique such as K-Means [23] to determine clusters of similar user sessions;

8. Analyze the clusters and classify them according to the probability associated with user action, concluding the user behavior analysis.

3.2 Validation

In order to validate the characterization model, we present our validation methodology, which is based on generating a synthetic log that mimics the behavior of a user. The characteristics of generated log is matched with the characteristics of the actual log so that we are able to verify the precision of the modeling process. The generation of latency-aware logs enables the enhancement of workload generators such as SURGE [9] or httperf [35], so that they generate Web workloads considering the interaction between the users and the system.

Current workload generators [9, 35] do not allow the simulation of user reactions and we believe that this innovation may provide significant improvement in the quality of the workload generated. Therefore, a major goal of the *USAR* validation strategy is to be the starting point for the construction of a simulator that mimics actual user reactions, based on the relation between IAT and latency.

To capture and replay the properties of the user behavior, we model and implement an user action simulator, parametrically defined, as shown in Figure 2, that deals with three types of data:

1. Input data: specify the characteristics of the users that interact with the system. The input data defines the types of users, their behaviors, the user profiles that compose each behavior, and the following distributions for each *USAR* level:

   - User: we define the probability distribution of user types, the probability distribution of behaviors that compose each type and the probability distribution of user profiles (groups obtained in step 6 of the characterization methodology) that compose each behavior.
   - Session: the probability distribution of session length is defined. This distribution will be used to determine the number of user actions in each user session.
   - Action: we determine the distribution of the user action classes in sequences that are directly related to the user profiles. As a result, we specify the popularity of the actions in each profile.
   - Request: the last level involves the generation of the requests that compose the Web workload. For a precise workload generation, we must determine the popularity of the objects and their size probability distribution. These distributions are not used in the generation of user action classes, but are important to transform these classes into real requests.

2. Execution parameter: parameter that are set in a per execution basis, such as number of sessions that must be simulated.

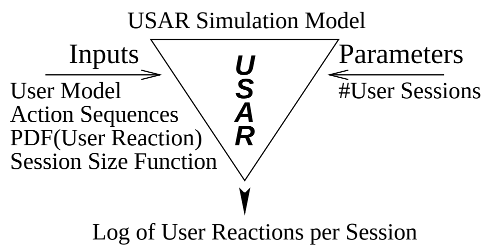


Fig. 2. The USAR Simulator

3. Output data: a log file composed of user sessions. Each session is composed of classes of user actions that simulates accurately the user behavior. The log file can be applied to a workload generator to produce Web workloads that are representative for evaluating the performance of the server and the network.

3.3 Applying USAR: A Real Case Study

This section presents a case study that demonstrates the effectiveness and applicability of the characterization model presented in previous subsections and its validation strategy. We focus on the innovative features of the model, more specifically the user level characterization. Further, we will just evaluate basic aspects and explain the main results related to the other levels.

Our characterization is based on one month data collection from the Squid proxy-cache server of the Federal University of Minas Gerais (UFMG). The log files obtained contains an entry to each request containing the following information: request timestamp, latency to obtain the resource, requester IP, object status in cache and HTTP status code, object size (bytes transferred), method (GET, POST), URL of the resource requested, peer status and host (for caches that participate in a hierarchy), and mime type.

We start by performing a detailed analysis of this workload in order to determine the diversity of the user population, the variety of Web sites being accessed, the latency variation that the users experience, among other information.

3.3.1 Workload Characterization Model

In this section we present the resulting characterization of our case study from the request to the user level. We used just four weeks in our characterization, since our initial analysis showed that the amount of information given by these four weeks is similar to the information given by the whole period.

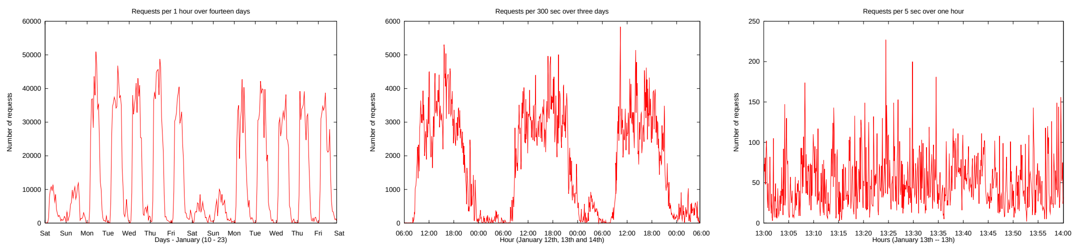
Request level characterization

In the request level characterization we focus on the requests regardless of the action, session, and user associated. The log L records the accesses of users from one of the biggest federal universities in Brazil and it has a considerable number of requests per day.

We apply the methodology at the request level, analyzing about 9 million requests issued from about 500 unique IP addresses statically assigned, generating a traffic of almost 90

Table 1. General information about the workload

	Week			
Attribute	**1**	**2**	**3**	**4**
# Requests ($x10^6$)	2,44	2,10	2,15	2,08
MegaBytes ($x10^3$)	34,4	19,9	16,8	20,7
# Unique IPs	488	485	497	507
# Sessions	6952	6979	7369	7756
# Unique Obj. ($x10^5$)	4,82	4,13	4,29	3,97



(a) Arrivals per hour over 2 weeks (b) 300s over 3 days (c) 5s over one hour

Fig. 3. Total number of requests arriving at proxy-cache at various time scales

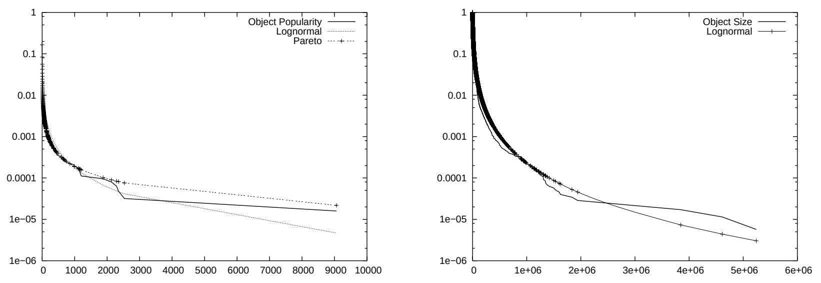
Gbytes. As expected, 98% of the requests are for HTTP objects and the remaining requests are either for FTP or HTTPS objects. Table 1 presents general information about the log, which contains more than two million requests per week, a significant number of unique objects, unique IPs, and user's sessions.

This analysis demonstrates that traditional characterization dimensions, such as object size, object popularity, and request arrival distribution, match previous characterizations of the same type of traffic [9, 34].

Figure 3 shows three graphs that plot the request arrival process at three different time scales, nominally, one hour, five minutes and five seconds.

The analysis of the graphs show clearly that the arrival process of requests presents a self-similar characteristic. A self-similar object is exactly or approximately similar to a part of itself (i.e. the whole has the same shape as one or more of the parts).

Figure 4 presents two graphs that show the probability distributions of objects and object sizes. In both cases, we can clearly see that both distributions are quite skewed, as observed in other characterizations of Web-related traffic.



(a) Popularity of Objects (b) Object Size

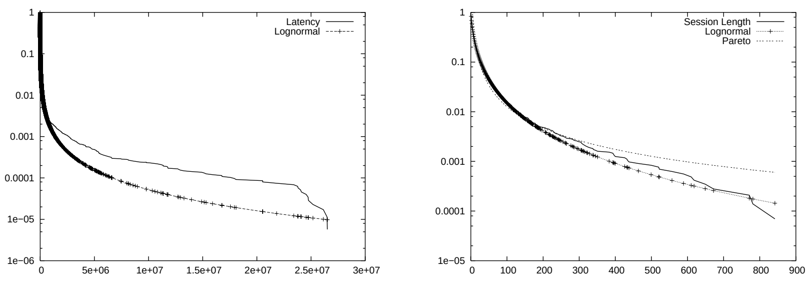
Fig. 4. Request Level Characterization

Action level characterization

The action level characterization demands the generation of a temporary log with the relevant requests, that is, requests that are directly associated with actions. The characterization consists of the identification of the action types, which are directly related to the functions provided by the application, and the multi-scale analysis of the actions, among other relevant analysis.

Since our analysis is based on a proxy log, it is not possible to determine the type of the user action by simply analyzing its URL and other data that composes the log, since we do not have contextual information about the service being provided. Therefore, in the context of our case study, the action types are not as relevant as for other application domains, such as an E-business service.

Considering only the requests to HTML objects, the total number of user actions is 160585. We modeled the probability distribution of the action latencies and found that it fits a log-normal distribution, as shown by graph in Figure 5(a). This fitting demonstrates the high variability of the observed latencies, which gives room to investigate the correlation between the server response time and the user reaction.



(a) Action Latency (b) Session Length

Fig. 5. Characterization - Action and Session Levels

Session level characterization

Considering just the requests to the 62770 unique HTML objects and using a threshold of 1800 seconds for session duration, we identified 14352 user sessions associated with 518 unique IP addresses. One initial step of the session level characterization is to generate a log Ls for each session s.

We then analyze the session length (Figure 5(b)), in terms of number of requests, and notice that more than 90% of the sessions are composed by at most 26 requests, the average and the maximum session lengths are 12.2 and 1105 requests, respectively. The analysis of the session duration shows that most of the sessions (almost 80%) last for less than 1800 seconds and the average session duration is 1172 seconds. We also observe a significant variation in the distribution of requests among sessions. Note that the observed average session length shows the suitability of the log to our work, since short sessions do not provide enough information to model user behavior.

User level characterization

This subsection describes the user level characterization performed for the case study according to the proposed methodology. The first step is to generate a temporary log Lu by putting together the sessions of each user u, although the session identifiers are kept.

We then analyze the user data from the following perspectives: IATs between consecutive requests, latency associated with user requests, IAT-latency ratio, and IAT-latency difference.

We discretized IAT and latency measures using functions that correlates them, more specifically the ratio (RAT) and the difference (DIF) between them. These metrics are defined as:

$$DIF(k) = I(k, k+1) - L(k), \ \forall \ k \in Lu;$$

$$RAT(k) = \begin{cases} \text{I(k,k+1)/L(k)} & , \text{DIF(k)} > 0 \\ \text{L(k)/I(k,k+1)} & , \text{DIF(k)} < 0 \\ 1 & , \text{DIF(k)} = 0 \end{cases} ;$$

where k is a user request, $I(k, k+1)$ is IAT between request k and $k+1$, and $L(k)$ is the latency associated to the request k.

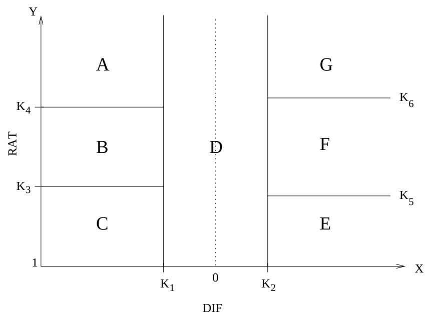


Fig. 6. Discretization Model

Figure 6 depicts the discretization model based on the two functions. The x axis is associated with the DIF function and the y axis with the RAT function. The model defines seven user action classes (A to G), using two limit values for each axis. Values k_1 and k_2 divide the positive and negative sides of DIF function, defining a zone close to zero, where we can not say much about the user behavior. This zone represents values of IAT and latency very close to each other, which can represent situations such as: users who request objects and ask another one few seconds before the first request answer arrives; and users who request objects and do not process the answer, since they request another object immediately after the request answer arrives. As shown in Figure 6, we define an interval D between k_1 and k_2, which comprises all values of RAT, motivated by the fact that the value of the RAT function does not affect it. Values k_3 and k_4 break the vertical scale in three different zones, according to RAT function that quantify the correlation between IAT and latency. We then define three classes (A, B and C) for DIF values that are less than k_1. These classes represent behaviors where users do not wait for the answer to their requests before asking another object. The same strategy is applied to the actions that have a DIF greater than k_2, that is, we define three classes (E, F and G) that represent behaviors where users wait for the answer to their requests before asking another one. The boundaries of these classes is defined by two other constants: k_5 and k_6. For instance, Class E represents actions where the user request a new object a short time after receiving the previous object. On the other hand, class C represents users who do not receive the object within the expected delay, but wait a significant amount of time, considering the object's latency. We plot the points' histogram and decide the values that divide the classes. (k_1 and k_2 are assigned to the values -0.1 and 1, respectively. The values of k_3 and k_4 are 2 and 4, respectively, for the discretization of classes A, B and C. For the delimitation of classes E, F, and G we choose the values 4 and 8 for k_5 and k_6).

We then transform user sessions into sequences of user action classes using this discretization strategy. This transformation is a direct map one-to-one from application of functions RAT and DIF to each request of user session, defining a user action class. As a result, we define, for each action, a pair $u(DIF((k), RAT(k))$ of user request, where k is the current request in the user session. This pair corresponds to a location in the discretization model, defining the user action class for each of his or her actions. The total number of user actions is 160585. Table 2 presents the frequency of user actions.

Table 2. Distribution of User Action Classes

User Action Classes(%)						
A	**B**	**C**	**D**	**E**	**F**	**G**
1.56	1.47	2.45	3.80	8.40	8.97	73.35

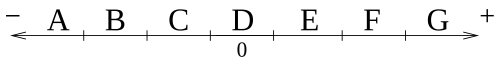


Fig. 7. User Profile Tendency - Patience Scale

Considering the scale presented in Figure 7, we define six user profile trends, each of them representing a user behavior trend in the patience scale, with the following characteristics.

Impatient: The sequence of actions are in the negative side of the scale, including zero (e.g., $C \rightarrow A \rightarrow B$, or $A \rightarrow C \rightarrow B$, or $C \rightarrow D \rightarrow B$, or $C \rightarrow C \rightarrow C$, or $A \rightarrow A \rightarrow A$). Represents a variation in the user behavior, keeping the impatient tendency.

Patient: The sequence of actions are in the positive side of the scale, including zero (e.g., $E \rightarrow G \rightarrow F$, or $G \rightarrow E \rightarrow F$, or $F \rightarrow D \rightarrow G$, or $E \rightarrow E \rightarrow E$, or $G \rightarrow G \rightarrow G$). Represents a variation in the user behavior, keeping the patient tendency.

Continuous: The sequence of actions shows low variability, staying in fixed class at zero (e.g., $D \rightarrow D \rightarrow D$). It represents a fixed tendency, situations where the latency of the requested object and the IAT are very close. This is a typical web robot behavior or users whose tendency is not well-defined.

Impatient tendency: The sequence of actions represents a impatience tendency (e.g., $G \rightarrow D \rightarrow A$, or $C \rightarrow B \rightarrow A$, or $G \rightarrow F \rightarrow E$, or $G \rightarrow A \rightarrow C$, or $E \rightarrow B \rightarrow C$, $F \rightarrow G \rightarrow A$, or $E \rightarrow G \rightarrow B$).

Patient tendency: The sequence of actions represents a patience tendencies that usually move right in the scale (e.g., $A \rightarrow D \rightarrow G$, or $A \rightarrow B \rightarrow C$, or $E \rightarrow F \rightarrow G$, or $B \rightarrow G \rightarrow F$, or $C \rightarrow F \rightarrow E$, or $B \rightarrow A \rightarrow E$, or $C \rightarrow B \rightarrow F$).

Irregular: The sequence of actions that move from the negative (positive) side to the positive (negative) side and return to the negative (positive) side or zero - and sequences that move from the negative side or zero to the positive side and return to the negative side. (e.g., $C \rightarrow E \rightarrow B$, or $A \rightarrow G \rightarrow D$).

We then translate the user session log into a sequence of user profiles. The new user session representation consists of sequences of actions, where each tendency fits to a patience scale variation.

We use a Java-based tool for machine learning and data mining [23], which implements regression, association rules and clustering techniques. More specifically, we use the algorithms *K-Means* (KM) and *Expectation Maximization* (EM) [28] to perform clustering.

We finally got the results with *K-means*, that show interesting conclusions related to the user behavior. Adopting within cluster sum of squared errors we identify 7 as the best configuration for the number of clusters. The distribution of user profiles for each cluster is presented in Table 3. It shows the identification of clusters, the percentage of user sessions and the distribution of user actions according to user profiles, in each cluster.

Table 3. Distribution of Sessions and User Profiles (clusters)

	Sess	**User Profile (%)**					
Id	**(%)**	**1**	**2**	**3**	**4**	**5**	**6**
1	50	0.01	99.34	0.01	0.21	0.37	0.06
2	17	0.57	67.69	0.86	14.92	10.14	5.81
3	3	31.41	0.48	33.77	0.88	6.17	27.28
4	4	0.2	0.6	0.0	97.63	1.48	0.09
5	11	1.28	36.28	0.56	41.65	13.86	6.37
6	11	0.27	50.04	0.14	6.96	40.16	3.44
7	4	0.0	0.03	0.13	1.22	98.11	0.24

Analyzing the clusters, we can describe them as:

Cluster 1: Almost all users clustered in this group presents a *Patient* profile during their sessions.

Cluster 2: Presents a significant occurrence of two profiles (*Patient Tendency* and *Impatient Tendency*), a little of *Irregular*, and the majority of *Patient* profile.

Cluster 3: Represents a balanced occurrence of three profiles (*Impatient*, *Continuous*, and *Irregular*).

Cluster 4: Almost all users clustered in this group presents a *Impatient Tendency* profile during their sessions.

Cluster 5: The profiles *Patient* and *Impatient Tendency* have been identified in this cluster as the most significant. Also there is a considerable amount of *Patient Tendency* in this group.

Cluster 6: The profiles *Patient* and *Patient Tendency* have been identified in this cluster. It is a group of typical patient users, that maintain a strong patient tendency under some variation.

Cluster 7: Almost all users clustered in this group presents a *Patient Tendency* profile.

The most popular cluster is number 1, which corresponds to half of the number of user sessions. The cluster number 2 has 17%, followed by clusters 5 and 6, both with 11%. The remaining 11% is divided by clusters 4 (4%), 7 (4%), and 3 (3%). Analyzing them we can observe the predominance of *Patient* profile, but the occurrence of *Patient Tendency* and *Impatient Tendency* is either very significant. The profiles *Impatient*, *Continuous*, and *Irregular* are significant in the cluster with the small popularity.

The characterization is complete and now we are going to validate it, as described in the next section.

3.3.2 Workload Characterization Validation

In this section we present the generation of user action classes, as a strategy for validating the characterization model and simulating the behavioral characteristics of the users of the proxy-cache server of the Federal University of Minas Gerais (UFMG). Analyzing the results obtained in the workload characterization (section 3.3.1), we defined the input data listed in section 3.2 as follows:

1. One user type, seven user behaviors represented by the clusters (Table 3) and six user profiles.
2. The probability distribution applied to the user behaviors use the values obtained in the clusters (second column of Table 3) to determine the percentage of occurrence of behaviors for the user.
3. The probability distribution associated with the user profiles (impatient, patient, continuous, impatient tendency, patient tendency, and irregular) uses the values listed in Table 3 to determine the percentage of occurrence of profiles in each user behavior.
4. The session length distribution, according to the number of user action classes per session, follows *log-normal* distribution (see Figure 5(b)) with $\sigma = 1.4631$ and $\zeta = 1.4303$. The least square computed for this function is 0.02266.
5. The probability distribution of user action classes in sequences uses the percentage of occurrence of each sequence in the real log - considering sequences of size 1, 2 and 3 - to determine weights for the further simulation. We explain how the sequences are composed later in this section.
6. The characterization shows that the popularity of objects follows *Pareto* distribution with $\alpha = 0.9803$ and $k = 0.1589$. To certify the quality of the distribution fit, we calculate the least square measure and obtained a value of 0.0006 for *Pareto*. The size of the objects are strictly correlated to the latency observed in the server and does not follow any known distribution (see Figure 4).

The characterization process defines seven user action classes that correlate IAT and latency. To map these classes to the six user profiles, we decide to group the action classes into sequences. Combining the seven classes we obtain 343 possible sequences of size three, 49 of size two and 7 of size one - sequences of size 2 and 1 are used as the last sequence in case of the session length not being multiple of 3. Each sequence has a weight, calculated according to the absolute value of its occurrence in the real log and the percentage of sessions where it appears. At last, the sequences are mapped to a user profile according to a tendency, as presented in section 3.3.1.

Using these data and considering the number of sessions informed as a parameter, the generation process, for each session, follows these steps:

- Choose a user behavior according to the distribution given as input data (for this case study a probabilistic distribution).
- Calculate the session length, number of user action classes, and the number of sequences that compose the session.
- Generate the sequences of user actions following the distribution of the user profiles that compose the chosen user behavior (see Table 3). Some sequences are more frequent than others considering the weight calculated for each sequence.

At the end of the generation process, the output data is a synthetic log containing sessions of user action classes that considers the Quality of Service of the system to simulate the real interaction of users with the proxy-cache server. We should emphasize that the generation of an accurate synthetic log is a challenge because we created a quite abstract model of a behavior that is often highly variable. As we see next, the precision achieved by the model is one of the contributions of this work.

The objective of the simulation is to show the applicability of the characterization model in order to generate a synthetic log that gives the user action in response to the quality of service provided. Using an efficient method for generating discrete random variables with general distributions [42], we simulate the distribution of the sessions among user behaviors accurately. Further, the simulation produces precise results in terms of the distribution of sequences over the six user profiles observed in each cluster (see Table 3). It is hard reproduce exactly the same sequences because the session length distribution was calculated for the whole log.

Nevertheless, a careful analysis shows a good result regarding the frequency of occurrence of each user action class in the synthetic log, when compared to the real one. Table 4 presents the results for our proxy log.

Table 4. Distribution of User Action Classes

	User Action Classes (%)						
Log	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Real	1.56	1.47	2.45	3.8	8.4	8.97	73.35
Synthetic	1.54	1.44	2.4	3.76	8.38	8.95	73.51

The simulation values for all user action classes are very close to the real log characterization and demonstrate the feasibility of the adoption of well-defined models, based on the user behavior, to characterize web services workloads, considering the server-side view of the user interaction under variable latencies.

Finally, during the validation, we identify some directions to improve this work, such as the analysis of the correlation of session length and user profile, which can minimize the difficulty of simulating the real distribution of the sequences among user profiles.

4 Reactive Workload Generation

This section discusses the problem of generating workloads with dynamic aspects, showing the main changes we have made to the *httperf* workload generator in order to provide reactive workloads.

The ability to generate a stream of HTTP requests that mimics a population of real users is important for performance evaluation and capacity planning of Web servers. However, generating representative Web reference traces is a challenging.

There are a number of workload generators (see Section 2), but they do not consider the impacts of response time in the way users react. They generate similar workloads despite of the response time perceived. One of the objectives of this section is to show that there are significant differences between the workload applied to a Web server that considers the impact of response time in user reaction.

The *USAR* Model [37] introduces new concepts that enable realistic modeling of user reactions to variations in the response time. To apply these concepts, some important characteristics are needed by a workload generator:

- It must be able to initiate a new request, even if the last one has not finished yet, i.e., the related answer has not arrived. This aspect is very important mainly when the response time grows because some users may present an impatient behavior, without waiting completely for the last answer.
- The inter-arrival time must not be static but may vary dynamically according to the response time perceived and the user action class of the *USAR* model, that associates to each burst a value that correlates the response time and the inter-arrival time.

Moreover, it is important to introduce the concepts of burst and session. Bursts consist of a sequence of requests for fetching a web page with embedded objects (pictures, for example). A burst is submitted to the server when a user clicks on a link or requests a Web page. Bursts mimic the typical browser behavior where a click causes the browser to first request the selected Web object and then its embedded objects. A session consists of a sequence of bursts in which the time between any two consecutive burst is below a certain threshold.

4.1 Workload Generation using httperf

In this work we use *httperf* as the tool for workload generation. We choose it because it provides an effective way of generating HTTP workloads and measuring performance. We create a modified/extended version of *httperf* in order to allow it to generate new workloads and make it compatible with the *USAR* Model. The next paragraphs describe the new features we have modeled and implemented.

httperf has a module called *wsesslog*, which submits requests based on a user session file. It contains many aspects of user sessions, such as the number and sequence of requests, HTTP method, think-time and *burst* length. In order to aggregate the reactivity model created by *USAR*, we have added information about the user action class to the user session structure.

In order to determine the user action class from patience scale of the *USAR* model, we need the value of the response time observed by the client (in this case, the *httperf* itself) and the client think-time. A typically *wsesslog* file contains the think-time, so we only have to obtain the response time.

The value of response time can be easily obtained in *httperf*, since it is built around the concept of events. For example, every time a session is created or destroyed or a request is started or finished, a new event is triggered. These events (see Figure 8) may be captured through call-back handles, defined using *httperf* API functions. There is a response time associated to each request and another one that belongs to the burst. Basically, the values of response time could be obtained using two events, one triggered by request start and the other one triggered when the response completely arrives. To obtain real values of response time, we submitted a workload file based on TPC-W [22, 33] using the original version of *httperf* to the test environment.

Figure 8 shows the traditional mechanism of executing an user session through the elements assigned with the flag 0. It represents the client and server sides and some events associated to the execution. The vertical space represents the time. The figure illustrates the session duration and the concepts of response time, think-time and IAT. Moreover the expected response time and the expected IAT are represented (shapes labeled with numbers 1 and 2). The main request of the burst is represented by a bold line and the embedded requests are single lines.

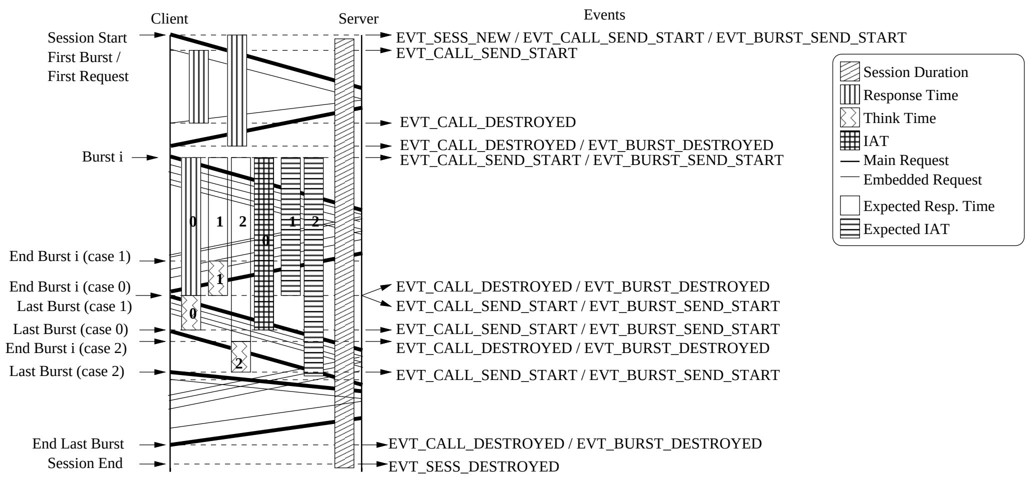

Client
Server
Events
Session Start
First Burst /
First Request
Burst i
End Burst i (case 1)
End Burst i (case 0)
Last Burst (case 1)
Last Burst (case 0)
End Burst i (case 2)
Last Burst (case 2)
End Last Burst
Session End
EVT_SESS_NEW / EVT_CALL_SEND_START / EVT_BURST_SEND_START
EVT_CALL_SEND_START
EVT_CALL_DESTROYED
EVT_CALL_DESTROYED / EVT_BURST_DESTROYED
EVT_CALL_SEND_START / EVT_BURST_SEND_START
EVT_CALL_DESTROYED / EVT_BURST_DESTROYED
EVT_CALL_SEND_START / EVT_BURST_SEND_START
EVT_CALL_SEND_START / EVT_BURST_SEND_START
EVT_CALL_DESTROYED / EVT_BURST_DESTROYED
EVT_CALL_SEND_START / EVT_BURST_SEND_START
EVT_CALL_DESTROYED / EVT_BURST_DESTROYED
EVT_SESS_DESTROYED
0
1
2
0
1
2
1
0
2
Session Duration
Response Time
Think Time
IAT
Main Request
Embedded Request
Expected Resp. Time
Expected IAT

Fig. 8. Client-server interaction mechanism in *httperf*

4.2 Reactive Workload Generation

This section explains how to generate the reactive workload using the user reactivity model. First, it is important to explain the concept of *user impatience*. Traditional workload generators model that a new request of the user must wait the last one to be completed before dispatching a new one. This approach does not allow to represent the situation where the user wants to send a new request even though the last one has not finished yet. Users would do this because the response time for the last request is unacceptable for him/her, for example. We name here *user impatience* this situation that can be modeled, demanding the ability of the workload generator to allow non-blocking sessions, i.e., a burst of requests may begin before the last burst has completed.

The original *httperf* does not consider *user impatience*, i.e., if a request takes a long time to complete, the program keeps waiting for it forever. The only *httperf*'s parameter close to this concept is the time-out value, but it is a radical solution: when a request times out, the *httperf* finishes the entire session.

In order to reproduce the *user impatience*, we changed the way *httperf* schedules the burst that is submitted for each session. The original implementation waits until the last submitted burst finishes to start a timer event that triggers the next burst. We adapted the *wsesslog* to start a timer event as soon as the first request of the burst is submitted. But how long should *httperf* wait before triggering a new burst? The value may be calculated using the user class and the response time of the former request.

We instrumented *httperf* to record some important events. We recorded the following values for each request and burst: session identifier (SESSID), time when it was sent (SNDREQ, SNDBUR) and received (RCVREQ, RCVBUR), response time (client perspective), bytes received, and if the request had timed out; and for each session created or destroyed, how many sessions were active (SESCNT).

As a result we create a new version of *httperf*: non-blocking and reactive. This version supports submitting requests that time-out after a period specified by the user think-time. This can be obtained during execution time, through these steps:

- Each session records its last response time observed, i.e., how long the last successfully submitted burst spent up to the end, and the burst size (in bytes). These two metrics are useful in order to estimate the next expected response time (ER) for the new burst, i.e., the amount of time that the user expects to be served.

- Each burst has associated to it its user class and the exact moment of time when the first request is issued.

- The Expected Inter-arrival Time (EIAT) can be computed considering the think-time (Z).

Figure 8 shows the reactive version of the workload generator by the shapes with the numbers 1 and 2. In this two examples the IAT is computed dynamically according to the server response time.

5 Evaluating the Impact of Reactivity

In this section, we present the results of our experimental study to analyze the impact of workloads that simulates the reaction of users to the Quality of Service provided.

5.1 Methodology

The simulation environment of our experiments is composed of a HTTP Server (Apache 2.0), an application server (Apache Tomcat), a relational database server (MySQL) and a client (*httperf*), each running on different machines. Each machine runs Linux with kernel version 2.4.25, having a Intel Pentium 4 1.80GHz CPU, and 1GB of main memory.

For best performance, we have turned off all unnecessary services and configured the operating system to support a number of file descriptors that was enough for our experiments (65000 file descriptors).

We adapted the workload generated based on TPC-W to add the information related to the user behavior, following the steps:

1. To obtain a synthetic workload that could follow the same navigation rules as would a real user, we create a base workload following TPC-W recommendations and its CBMG (from *Custom Behavior Model Graph*, [30]). The workload generated, *wl-tpcw*, is composed of 5000 user sessions with medium session length of 124 bursts.

2. We convert the *wl-tpcw* workload on a new one, *wl-httperf*, which is compatible to the format used by the *httperf*'s module *wsesslog* [35].

3. We submit the workload *wl-httperf* to our simulation environment using the original version of *httperf* and record the real response times under the client's perspective.

4. With the recorded response times and the workload *wl-httperf*, we apply the *USAR* characterization model, obtaining the distribution of user actions for each burst of requests.

5. We add to the workload *wl-httperf* the information obtained in the last step, obtaining the workload *wl-httperf-react* that can be used by the new version of *httperf* to generate workloads with reactivity.

Our main objective with these experiments is to analyze the impact of reactive and non-reactive workload in the performance of servers. The reactive workload models users who act dynamically according to variations in the response time.

It is important to emphasize that the number of simultaneous users is defined by the number of active sessions over the experiment time.

We execute experiments with many different workload configurations. In order to evaluate different types of load, we present three of them:

- A: a workload with 100 sessions with a rate of 100 sessions initiated per second.
- B: a workload with 1000 sessions with a rate of 100 sessions initiated per second.
- C: a workload with 5000 sessions with a rate of 100 sessions initiated per second.

For each configuration, we have the reactive an non-reactive approaches. The experiments evaluate a set of metrics for each scenario:

- Throughput: in our experiments we show both the output and input throughput. The latter corresponds to the rate of requests submitted per unit of time to the server. The former represents the rate of responses received by the client per unit of time.

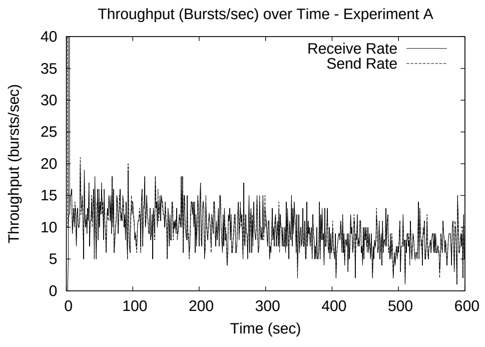
- Response Time: refer to the user perceived response time, consisting of the time between the submission of the request and the time when the client finishes to receive the response. It is important to explain the concept of response time, that is directly related to the way user reacts. Response time is a critical factor to users of interactive systems [30]. It is evident that user satisfaction increases as response time shortens. Modest variations around the average response time are acceptable, but large variations may affect user behavior.

- Active bursts: this information capture the number of bursts requested to the server but not yet responded, for each period of time.
- Active sessions: this information represents the number of sessions initiated but not yet finished, i.e., that have bursts that are active or have not been submitted yet.

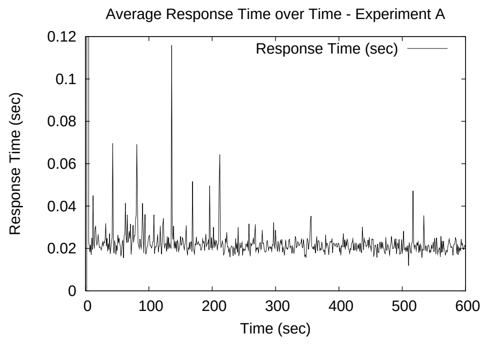
5.2 Results

5.2.1 Workload A Experiments

Figures 9 and 10 present the throughput (a) and response time (b) for the non-reactive and reactive experiments using workload A, respectively.



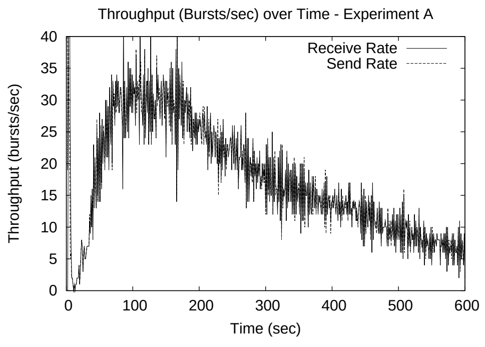
(a) Throughput



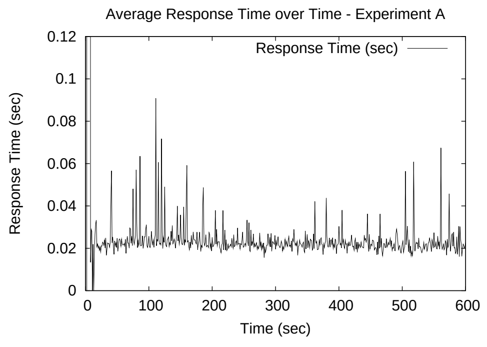
(b) Response Time

Fig. 9. Experiment A - Non-reactive - Bursts

The non-reactive experiment A achieves nearly 6000 bursts, with an average throughput of 9.2 bursts/second. The average response time is around 0.027 seconds. The average response time is very small, near zero (instantaneously). This confirms the non-overloaded state.



(a) Throughput



(b) Response Time

Fig. 10. Experiment A - Reactive - Bursts

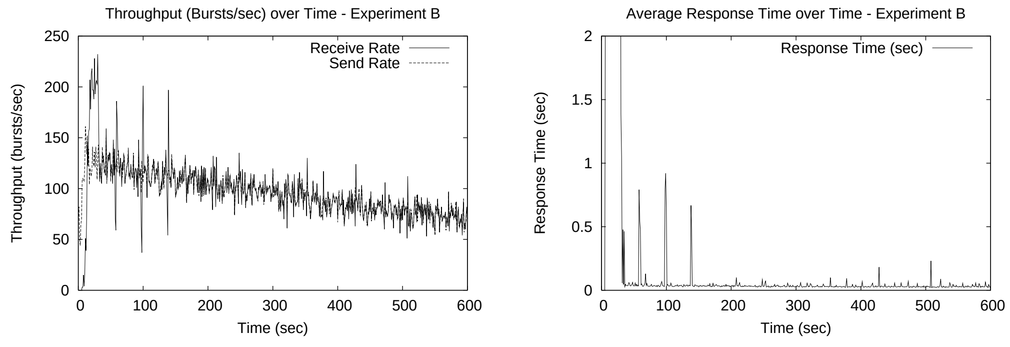
The reactive experiment A achieves more than 10000 bursts, with an average throughput of 16.1 bursts/second. The average response time is around 0.039 seconds. We can conclude that throughput raises without changing the response time, achieving a better performance. The average response time is very small, as observed by the non-reactive approach.

The number of active bursts during the non-reactive experiment A is very low, since the server is not overloaded. During this experiment, 85% of sessions ended, showing that reactivity allows users to reduce the estimated session time once the response time to their bursts of requests are very small.

The experiments for workload A show the server achieves a very good performance, guaranteeing that users perceives an instantaneous answer to their bursts of requests. A good response time rate allows users from the reactive experiment to increase the burst rate of requests. The increasing in the throughput rate without changing the response time rate shows the server is not overloaded. The decreasing in the execution time causes the reactive experiment to finish more sessions than the non-reactive one.

5.2.2 Workload B Experiments

Figures 11 and 12 present the throughput (a) and response time (b) for the non-reactive and reactive experiments using workload B, respectively.



(a) Throughput (b) Response Time

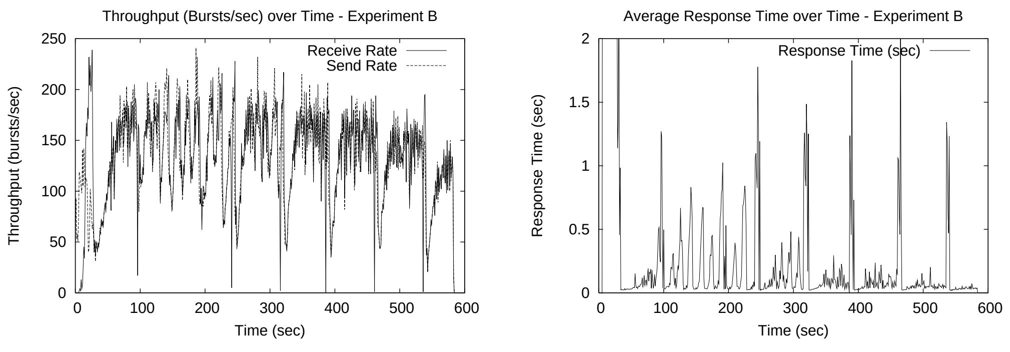
Fig. 11. Experiment B - Non-reactive - Bursts

The non-reactive experiment B achieves 57000 bursts, with an average throughput of 92.2 bursts/second. The average response time is around 0.24 seconds. The average response time is very small, near zero (instantaneously) with peaks under 1 second. This confirms the non-overloaded state.

The number of active bursts during the non-reactive experiment B presents a stable behavior, once there are not problems with performance. There are few peaks, that can be explained comparing it to the behavior of the response time. These peaks occur exactly when the response time presents some delay. During this experiment 45% of the sessions ended, the same percentage of the number of sessions that has ended in the non-reactive experiment A.

The reactive experiment B achieves 78000 bursts, with an average throughput of 114.4 bursts/second. The average response time is around 0.35 seconds. The average response time is still small, but not instantaneously as the first experiment(A). There are response time peaks under 2 seconds, but isolated situations that not endanger the performance of the server. In this few situations, the users may observe a small delay in the server response.

The number of active bursts during the reactive experiment B presents variations that can be explained comparing them to the response time behavior of this experiment. These



(a) Throughput (b) Response Time

Fig. 12. Experiment B - Reactive - Bursts

peaks occur exactly when the response time presents some delay, as expected. During this experiment, 90% of sessions finished.

It is interesting to note that the throughput rate of the reactive experiment B decreases exactly when response time rates raise, but in this case the change in the users reaction causes the throughput rate to raise again after a short time. We observe the application server keeps a very good response rate to the requests (around 1200/sec) without overload.

5.2.3 Workload C Experiments

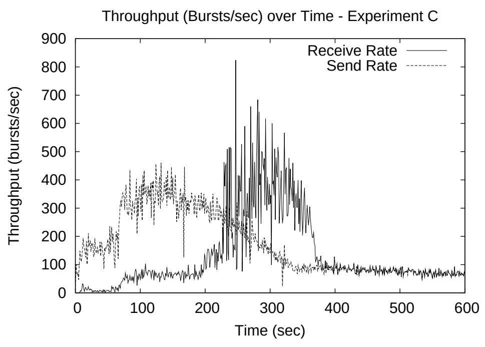
Figures 13 and 15 present the throughput (a) and response time (b) for the non-reactive and reactive experiments using workload C, respectively. In the same way, Figures 14 and 16 show the active bursts (a) and active sessions (b).

The non-reactive experiment C executes 80000 bursts, with an average throughput of 123 bursts/second, varying from 100 to 400 bursts/second after the initial seconds. The response time raises from few seconds to more than 120 seconds, with an average time of 40.7 seconds.

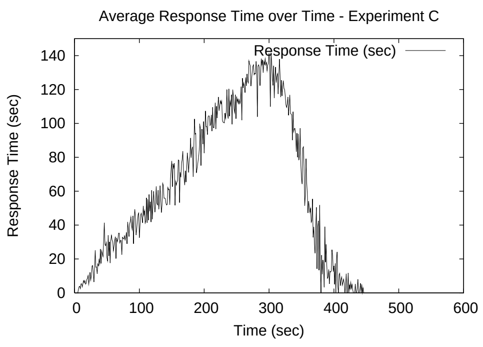
Considering the requests, the non-reactive experiment C achieves 580000 requests, with a throughput varying from 200 to 1600 requests/second. This amount of request is only 20% higher than the amount observed in experiment B. It is easy to observe that the server became overloaded. After 30 seconds from the beginning, the response time has already achieved the 10-second limit.

It is important to analyze what happened near 360 seconds. The following aspects are registered: the response time begins to decrease; the throughput decreases; the number of active bursts stabilizes; and the number of active sessions decreases fast. A detailed investigation shows the cause of this anomaly: the TCP/IP connection has timed-out, represented by the system error number 110 in Linux operating system. This problem has caused the following behaviors in the experiment metrics:

- Cumulative throughput: the difference between send and receive rates is more than 20000 bursts. This occurs because these amount of bursts do not finished as connections have closed after the TCP/IP time-out.
- Throughput - Figure 13(a): the throughput rate decreases and keeps near 100 bursts/second, consequence of bursts of a small number of sessions that stay active after the problem.



(a) Throughput

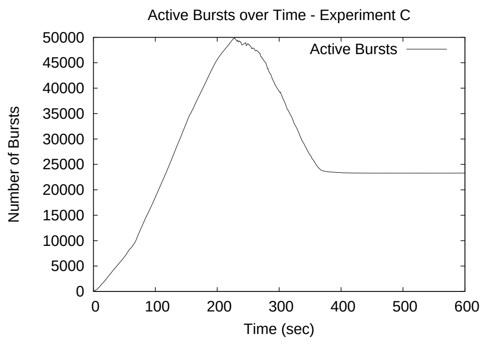


(b) Response Time

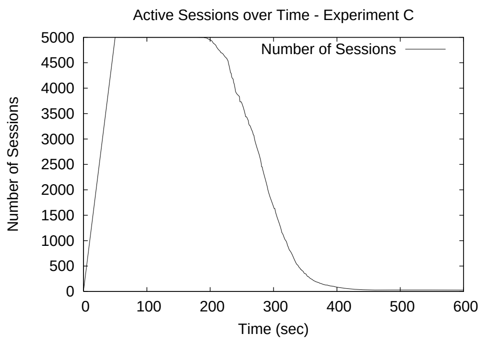
Fig. 13. Experiment C - Non-reactive - Bursts

- Response time - Figure 13(b): the average response time decreases fast when the problem with TCP/IP occurs. After 450 seconds, it achieves acceptable values by users.
- Active bursts - Figure 14(a): the number of active bursts continues high, as many bursts have not finished in consequence of the time-out of TCP/IP. After 400 seconds the value becomes balanced, with vary small variation.
- Active sessions - Figure 14(b): the number of active session decreases fast, which demonstrates that a lot of sessions begin to fail in consequence of the error identified. When the workload generator tries to open or to send requests and the TCP returns the error, the current session fails and close after there are no more connections available for it. Only an amount of 100 sessions become active after 400 seconds, representing the users who generate load to server from this point to the end of the experiment.

In this non-reactive experiment C we identify a big overload in the server. The response time values observed are unacceptable. Moreover the unavailability of the server represents one of the most serious problems that overload may cause - around 80% of the users stay waiting for server's answer without success.



(a) Active Bursts



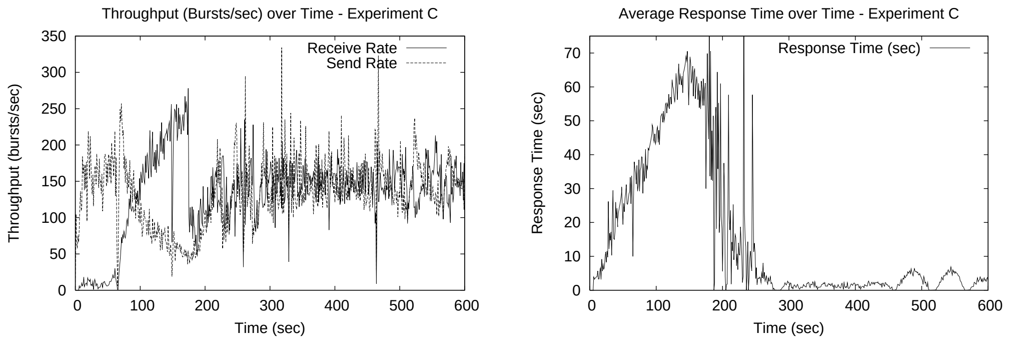
(b) Active Sessions

Fig. 14. Experiment C - Non-reactive

In the other hand, the reactive experiment C executes 80000 bursts, with an average throughput of 133 bursts/second, varying from 25 to 250 bursts/second after the initial seconds. The response time raises from few seconds to more than 60 seconds, with an average time of 13.7 seconds.

Considering the requests, the reactive experiment C achieves 690000 requests, with a throughput varying from 50 to 1800 requests/second. This amount of request is only 6% higher than the amount observed in experiment B. It is clear that the server became overloaded. After 20 seconds, from the beginning the response time has already achieved the 10-second limit.

In Figure 15(a), we can see the receive rate increasing and send rate decreasing from the period between 100 and 200 seconds. It is possible to observe a correlation between average response time and active bursts, as expected. The change in the way users react when the server is overload causes a delay in the session duration of the users, as can be seen in Figure 16 - 75% of sessions are still active after the experimental time.



(a) Throughput (b) Response Time

Fig. 15. Experiment C - Reactive - Bursts

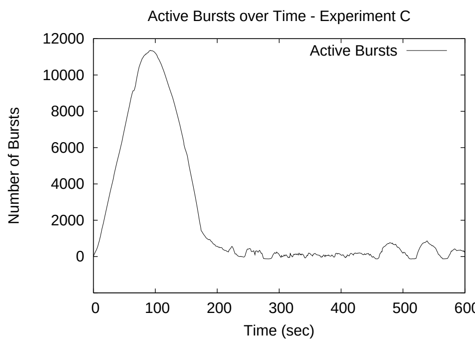
The non-reactive and reactive experiments present different scenarios. The first one has caused a heavy overload in the server, that keeps it unavailable for most part of the users. The reactive one has overloaded the server, but the reaction of users to unacceptable response time values changes their global behavior, allowing the server to save resources and later, to answer requests in a acceptable response time.

Analysing the two versions of workloads, the non-reactive and reactive ones, we observe that they cause different situations in all scenarios. This result is interesting, once it can be the base for research in QoS techniques that can consider the influence of user reaction in the performance of servers.

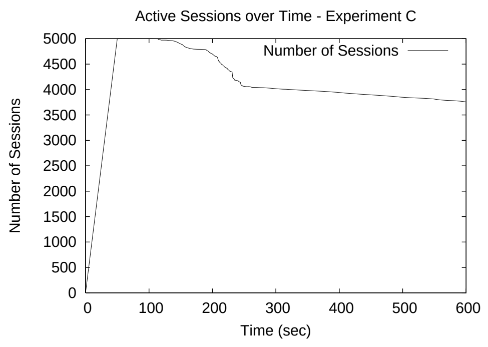
5.3 Discussion

The results of the experiments presented in this section show the changes in the performance of the Web server when the workload with reactivity is applied to it. The graphics show changes both in the values of each variable in time as in the form of each curve. Both the throughput and response time values have significant variations.

Analyzing the results we can conclude that the actual models of workload generation are limited, since they do not consider the user reactivity as an element of their workload models.



(a) Active Bursts



(b) Active Sessions

Fig. 16. Experiment C - Reactive

This indicates that models should be improved. In the same way, the actual models of Web server QoS have to be improved since they do not consider the user reactivity in their models also.

Moreover, the results demonstrate how important is to consider the correlation between user and server sides, once it can decrease the gap between the real and model scenarios. In the experiment A, users demanded a higher throughput from the Web server, that answers them with a very good response time. The experiment B presents another case, where the reactivity raises the throughput and the response time of the system, but it has achieved this in a balanced way. The experiment C has shown an interesting situation for the non-reactivity scenario, where a heavy workload has broken down the system. Adopting traditional workload generation mechanisms, the unavailability of the system was an expected situation, once the changing in the way users react is not considered. Our new model presents a completely different result, demonstrating the importance of a better understanding of the user-server interactivity process. These results can suggest new improvements in the overload control strategies.

6 Reactivity and QoS

Based on the models and conclusions discussed on the last sections, we explore the possibilities of using the reactivity knowledge to improve the Quality of Service offered by Web servers.

6.1 Reactive Behavior Analysis

The discretization model of the reactive behavior provides seven user reaction classes. Each user reaction class represent a different behavior that can be observed analysing the relation between IAT and the response time. Table 5 presents a representation of the functions RAT and DIF from the discretization model and the possible relation between IAT and the response time compared to the other classes.

The DIF function divides the values using two constants k_1 and k_2. It is used to separate the patient from the impatient behavior, since the difference between IAT and the response time (R) should be a positive value for patient classes and negative for the other ones. The RAT function intends to separate the patient classes and impatient ones in subclasses that permit a differentiation between bursts with similar behavior. RAT uses the constants k_3 and

k_4 to separate the impatient side and k_5 and k_6 the patient side.

Observing the RAT and DIF functions behavior for each class we can infer the typical relation between them. We identify that class A have the biggest RAT value among the other impatient classes (B and C). In the table we represent this by a $<<$ symbol. For classes B and C the IAT value is still lower than the response time but their RAT value is in general lower than the one for user class A. We represent the typical relation for classes B and C using a $<$ symbol. The same applies to classes E, F and G. G class has the greater RAT value compared to the classes E and F. This is why we represent the relation between IAT and response time with a $>>$ symbol for class G and with a $>$ for class E and F. We use a $\approx$ symbol for class D because the IAT and response time have close values.

Table 5. Relation between IAT and response time for each user reaction class

User Class	**DIF Function**	**RAT Function**	Typical relation
A	IAT - R $< k_1$	R / IAT $> k_4$	IAT $<<$ R
B	IAT - R $< k_1$	$k_3 <$ R / IAT $< k_4$	IAT $<$ R
C	IAT - R $< k_1$	R / IAT $< k_3$	IAT $<$ R
D	$k_1 <$ IAT - R $< k_2$	-	IAT $\approx$ R
E	IAT - R $> k_2$	IAT / R $< k_5$	IAT $>$ R
F	IAT - R $> k_2$	$k_5 <$ IAT / R $< k_6$	IAT $>$ R
G	IAT - R $> k_2$	IAT / R $> k_6$	IAT $>>$ R

In order to understand the behavior of each user reaction class we represent a typical request-response scenario on Figure 17. For each situation we represent a client issuing a request to the server that answers it according to its load. We represent different load scenarios for sake of better explaining the possible impact of each class behavior on the performance of the server. We represent a non-overloaded scenario, where the server takes less than 5 seconds to answer most of the requests. We represent also an overloaded scenario, where the response time grows achieving values greater than 5 seconds for most of the requests. It is important to notice that it is necessary to analyze more than just the response time in order to verify that a server is overloaded. In fact, if the server is overloaded the response time achieves high values most of the time.

Clients acting as impatient ones behave according to classes A, B and C. The figure presents their typical behavior. We observe that the IAT is lower than the response time. In a non-overloaded scenario their difference is not as significant as in overloaded ones. In overloaded scenarios the server will take more time to answer and the impatience of the client will make him ask more requests before receiving its previews one. Then in the point of view of a server, impatient users tend to submit more and more requests before receiving its previews ones. This can lead to an increase in the probability of an overload.

Clients acting as patient ones behave according to classes E, F and G. Their typical behavior as represented have the IAT greater than the response time, meaning that for each request submitted to the server the user waits its response before asking the next request. After receiving the response, an user acting as patient takes a time to proceed the next request. In overload situations, patient users tend to wait for the next request and takes a time to proceed. This is very important since the overload situation for the server may not increase due to the patient clients behavior. Class G presents the more patient behavior since its IAT tends to be greater than the ones for classes E and F.

In the point of view of the server the reactions of users provoke different changes in terms

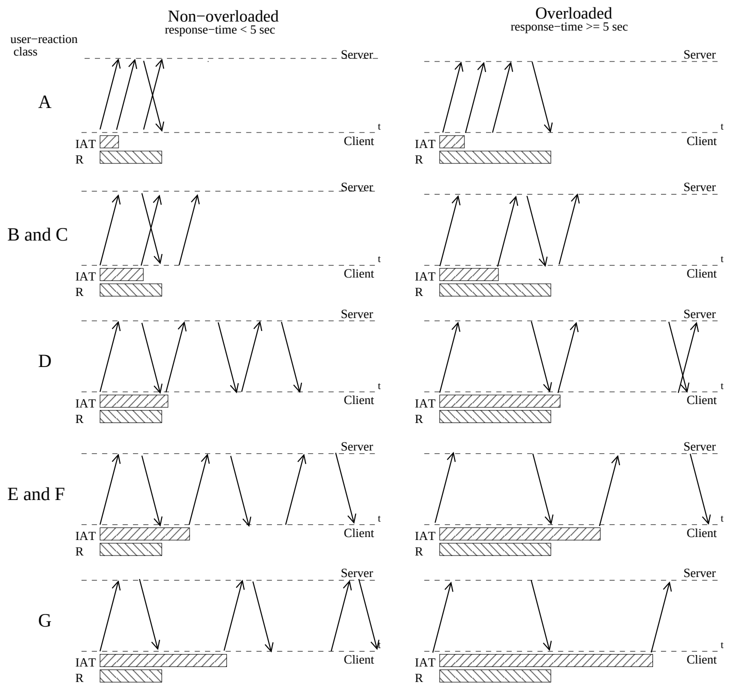


Fig. 17. Client Reactive Behavior

of load since the variations in the response time affect the rate of requests submitted. In fact, the impatient behavior tends to provoke an increase in the load over the server, since users behaving according to classes A, B and C asks requests in a higher rate most of the time. The patient behavior tends to make the load of the server to decrease due to the behavior of users reacting according to classes E, F and G.

Otherwise, in a real scenario the number of users behaving according to each user class is variable and the task of understanding its impact on the performance of a server is not obvious due to the complexity of such scenario. In the next sections we address this task by experimenting a web server with an reactive workload and by simulating a real web application.

6.1.1 Online Identification of User Behavior

In order to improve QoS through the use of reactivity, a web system must be able to identify the user behavior at each moment. This is necessary because the users may present different behaviors during their sessions. Since the policies we present in this session are based on the user action classes from the *USAR* model, we need to be able to aply the model for each user session in a way the server may present different responses to the users in terms of the QoS provided.

For sake of determining the user behavior on each session, we propose a user behavior classifier based on the reactivity model. The classifier identifies the user class at each moment, applying the RAT and DIF functions for each burst the server receives. The classifier calculates the latency presented by the server to answer the burst and the inter-arrival time between the current and the next burst. Then, it applies the RAT and DIF functions of the *USAR* reactivity model, determining by the discretization methodology, the user reaction class UAC for that burst.This methodology makes possible to obtain the user behavior at each period.

In order to improve the online classification, we propose a metric named USP, from user session profile, that represents the average user reaction class for bursts already been served for a user session. The following function calculates the USP for burst k:

$$USP(k) = \begin{cases} 4, k = 1, \\ \frac{UAC_{k-1}+USP(k-1)}{2}, k > 1, \end{cases}$$

$\forall\ k \in$ workload, where the USP for request 1 is equal to the average user action class D represented by the number 4. This function makes possible to obtain the mean user action class for a user session. We use it in the session admission control policy presented in Section 6.2.3.

6.1.2 Reactivity and QoS

The impact of reactive workloads on the performance of Internet services is discussed in Section 5, where we adopt the *USAR* model to evaluate the performance of a Web application and present a new version of the workload generator *httperf* [35], capable of reproducing the user reactivity. We performed experiments using an actual Web server using three distinct scenarios of load, and analyzing the following metrics: server throughput, response time, rate of submitted requests and number of active sessions. The results show that reactive workloads behave differently when compared to non-reactive workloads submitted to the Web server. The experiments demonstrate that different response times affect the workload, changing the rate of requests submitted and, consequently, changing the load of the server. Therefore, it is shown that server-side affects the client-side behavior, and vice-versa, proving the importance of the reactivity.

The main consequence of demonstrating the fact that reactivity impacts the server performance is that the actual models of Web server performance may be improved since they do not consider it. This motivates the investigation of new QoS strategies that may even mitigate negative effects of the reactivity and reinforce the positive ones.

6.2 QoS Improvement

In this section we present new QoS strategies that consider reactivity. First it is important to introduce the concept of burst, since our policies are based on it. Bursts consist of sequences of requests for fetching a web page and its embedded objects (like pictures). A burst is submitted to the server when a user clicks on a link or requests a Web page during its session. Bursts mimic the typical browser behavior where a click causes the browser to first request the selected Web object and then its embedded objects. Burst is a synonym for the term action we referred on the *USAR* model. A session consists of a sequence of bursts in which the time between any two consecutive bursts is below a certain threshold.

The proposed strategies are based on admission control and scheduling strategies. The basic idea is that bursts must be classified into user reaction classes, based on the reactivity model described in Section 3 that establishes seven user classes from A to G. The main novelty of this paper is the use of the reactivity to improve QoS and the idea to combine the proposed admission control and scheduling techniques.

6.2.1 Admission Control

Admission control rejects requests whenever the arrival rate is too high and it is necessary to maintain an acceptable load in the system. Without admission control, the response time increases when the system saturates. Traditionally, server utilization or queue length are criteria used in admission control schemes. This section presents briefly the approaches to implement the reactive admission control presented in [38].

6.2.2 Burst-Based Approach

Based on the $USAR$ model, this approach considers how users tend to react according to Internet service's performance. The burst-based policy rejects bursts when it identifies a fulfilled rejection rule. Traditional policies adopt just one limit to response time and begin to reject bursts once this limit is achieved. We define the policy as a function of the response time of a service (R) according to the following rules:

- $\alpha \leq R < \beta$: reject bursts of user action classes A, B and C.
- $\beta \leq R < \theta$: reject bursts of user action classes A, B, C and D.
- $R \geq \theta$: reject bursts of all user action classes.

In these rules, α, β and θ are values determined based on empirical results and criteria defined in the literature [30].

The idea of this policy is that users who have more impatient profile tend to react faster (that is, reload or submit a different request) than other users when the server presents high response times, degrading server's performance. The burst-based policy has a multiple criteria rule, minimizing the rejection impact, once less users may have bursts refused by the QoS admission control policy. In summary, this policy has the premise that, under overload scenarios, it may be better to give priority to users who have more chance to wait for the response.

6.2.3 Session-Based Approach

The session-based admission control policy rejects user sessions. Traditional policies employ a single response time threshold and start to reject all user sessions once this limit is reached. The burst-based admission control policy may affect all users, but the session-based policy is different, since it tends to affect fewer users. Considering this, reactivity may be important to identify which user sessions have to be dropped.

We monitor the average response time (R) and the user session profile for each session (USP), i.e., the average user reaction class of each burst that has already been served (see Section 6.1.1). We define the following three criteria:

- $\alpha \leq R < \beta$: reject user sessions with $USP < 4$, i.e., sessions associated with user reaction classes A, B and C.
- $\beta \leq R < \theta$: reject user sessions with $USP < 5$, i.e., sessions associated with user reaction classes A, B, C, and D.

- $R \geq \theta$: reject all user sessions.

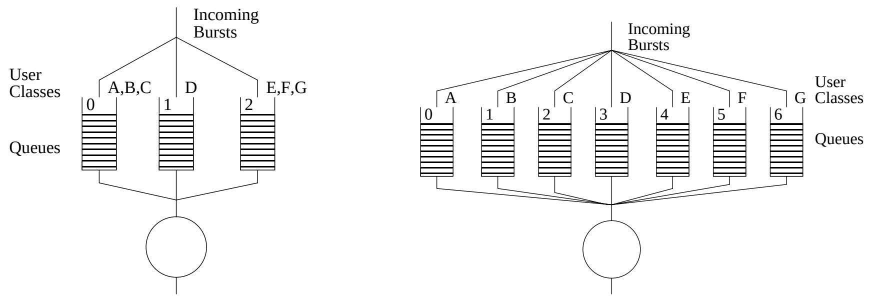
This policy is based on the same idea presented in the reactive burst-based policy, however applied to session-based control mechanism. The next subsection describes scheduling.

6.2.4 Scheduling

Most existing web servers provide services based on the Best Effort model, using FIFO scheduling [43]. As presented in Section 2, there are works that propose different approaches to improve the basic FIFO scheduling approach. They present gains compared to the Best Effort approach, but they fail to consider the reactivity, an important dimension of the user interaction with the service.

6.2.5 PFIN Scheduling Approach

In Patient-First Impatient-Next (PFIN) approach, bursts classified with impatient classes (A, B, and C) go to the low priority queues, and those with patient classes go to the high priority ones.



(a) Three Priority Queues (b) Seven Priority Queues
Fig. 18. Servers implementing PFIN Scheduling Approach

Figure 18 presents two variations of the PFIN approach, which differs in terms of the number of priority queues used to schedule the bursts. In (a) there are three queues. All the bursts identified with impatient classes are scheduled to the low priority queue and those identified with patient classes are scheduled to the high priority one. Bursts identified with classes A, B or C go to queue 0, bursts classified with D go to queue 1, and finally, the ones classified with classes E, F and G go to queue 2. In (b) we employ seven queues, one for each user class. The most patient classes get the high priority queues.

The PFIN policy is based on the idea that when the load is increasing the users who have more patient profiles tend to present a lower load to the server than the impatient ones since after receiving its response, they take more time to proceed and submit another request. Users with impatient profiles tend to react faster, asking requests even before receiving the previous one. This policy has the premise that under overload scenarios it may be better to give priority to users who have more chance to spend more time to submit requests, slowing down the server load and increasing the user satisfaction.

6.2.6 IFPN Scheduling Approach

In Impatient-First Patient-Next (IFPN) approach, requests classified with impatient classes (A, B, and C) go to the high priority queues, and those with patient classes (E, F, and G) go to the low ones. Like PFIN approach, two variations are proposed with three or seven priority queues.

The IFPN policy is based on the idea that when the load is increasing it is better to answer first the impatient users, in order to increase their satisfaction. The advantage of delaying answers to patient instead of impatient users is the fact that the satisfaction of patient users tends to take more time to degrade. In summary, this policy has the premise that it may be better to give less priority to users that have more chance to wait for the response to their requests.

6.2.7 Admission Control and Scheduling

We propose a hybrid three-level approach, that combines the two admission control strategies and scheduling. The idea of this new approach is to put together the advantages of each one. Admission control is good to avoid raising the response time to unacceptable values [38]. Scheduling is adopted to control the burst's priority according to the user class, providing a reduction in the burst's expiration rate, measure related to user satisfaction as we may observe on Section 6.2.7.

In the session-based approach, rejection of sessions is drastically started as response time grows. In the two-level approach, first of all, burst rejection is started, before the rejection of sessions. This strategy smooths the session rejection through a previous step. Once the burst rejection is not effective to slow down the response time, session rejection is activated. In Parallel, scheduling produces a burst reordering process that can reduce the number of unsatisfied users. We define the following criteria:

- $\alpha_1 \leq R < \beta_1$: reject bursts of user action classes A, B and C.
- $\beta_1 \leq R < \theta_1$: reject bursts of user action classes A, B, C and D.
- $R \geq \theta_1$: reject bursts of all user action classes.
- $\alpha_2 \leq R < \beta_2$: reject user sessions with $USP < 4$, i.e., with average user reaction classes A, B or C.
- $\beta_2 \leq R < \theta_2$: reject user sessions with $USP < 5$, i.e., with average user reaction classes A, B, C, or D.
- $R \geq \theta_2$: reject all user sessions.
- $R > \gamma$: turn on the scheduling policy.

This strategy rejects both bursts and sessions, but according to different limit values, balancing the burst's rejection and providing a way to schedule the bursts in order to raise the user satisfaction.

6.3 Experimental Evaluation

This section presents our experimental analysis of the new QoS policies. Section 6.3.1 briefly describes the simulator used in our experiments. Section 6.3.2 explains the methodology to evaluate the new QoS policies. Section 6.3.3 shows the experiments and results. Finally, Section 6.3.4 presents a discussion about the experimental evaluation.

6.3.1 Simulating Reactivity and QoS Policies

In order to evaluate the new scheduling strategies we extended the *USAR-QoS* simulator presented in [38] with novel features, capable of reproducing the behavior of different scheduling mechanisms. The architecture of the simulator is based on a real web application composed of a server providing a certain service and a set of clients. The server is composed of one or more queues and a processing unit with a certain limited processing capacity. The clients behavior is based on the reactive version of the *httperf* workload generator presented in [39].

The *USAR-QoS*' architecture is based on events and respects modularity. It uses the *Simpack Toolkit* [21]. Figure 19 presents a state diagram that represents *USAR-QoS*. For more details about the simulator, see [38].

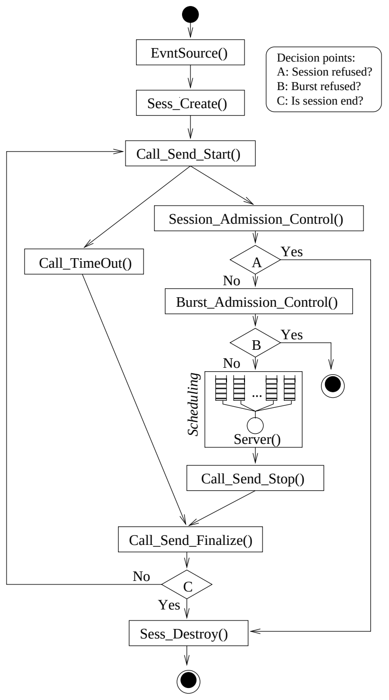


Fig. 19. State diagram representing the *USAR-QoS* simulator

6.3.2 Methodology

In order to evaluate the effectiveness of the proposed reactive QoS policies, we simulate them using the *USAR-QoS* simulator. We prepare a TPC-W-based synthetic workload trace file to be used as the input for the simulation.

We simulate several scenarios using different *USAR-QoS* configurations to observe how the application server behaves. We discuss in this paper a scenario in which the server achieves a high throughput and the response times observed raises over the user satisfaction threshold. We base this threshold on [30] where the authors identify three groups regarding the response time of a system:

- 0.1 sec: the limit when a user perceives that the system is reacting instantaneously.
- 1.0 sec: the limit when the flow of thought of a user is not interrupted, although the user may notice the delay.
- 10.0 sec: the limit when a user loses attention and the interaction with the system is disrupted.

Based on these values we implement each of the proposed reactive admission control and scheduling strategies in the *USAR-QoS*. As explained in Section 4, each reactive QoS policy has a set of values (α, β, θ and γ) that define its functioning. These values should be carefully chosen since the effectiveness of each policy depends on them. The values we choose are: $\alpha_1 = 3.0$, $\beta_1 = 5.0$, $\theta_1 = 7.0$, $\alpha_2 = 5.0$, $\beta_2 = 7.0$, $\theta_2 = 9.0$, and $\gamma = 0.0$.

We also evaluate the basic non-reactive QoS strategies on *USAR-QoS*. We implement a traditional session and burst admission control mechanisms with a threshold of 9.0 and 7.0 seconds, respectively. We implement also the Best Effort FIFO scheduling approach.

The experiments we present here consist of 5000 sessions, created in an average rate of 10 sessions per second. The server is configured to support 50 bursts per second of throughput. The trace file is based on the TPC-W benchmark [2, 33]. Each burst is identified with a different user reaction class according to the *USAR* model [37]. The overall distribution of action classes for the trace file is shown by Table 6.

Table 6. User reaction classes distribution for the workload

A	**B**	**C**	**D**	**E**	**F**	**G**
5%	10%	10%	15%	15%	15%	30%

We evaluate all combinations of reactive and non-reactive admission control and scheduling policies to compare the results and verify the most effective ones. Due to space constraints we show only the most relevant results.

6.3.3 Results

Tables 7 and 8 present the summary of the experimental simulation results. The results we have chosen to present here are the ones who better represent the benefits of considering reactivity for Web performance modeling. Each experiment is identified by a number from 1 to 5. The columns admission control and scheduling describes the experiment configuration in terms of the QoS strategies. The tables present the following information: session and burst admission control policy, scheduling policy, total experimental duration, higher response time value (represented as R), mean response time, total number of requests, responses and expirations of bursts, burst lost rate, and number of rejected sessions by the session admission control. The burst lost rate represents the percentage of bursts expired compared to the whole number of bursts requested to the server. It corresponds to the number of occurrences the user asks the next burst before receiving the previous response.

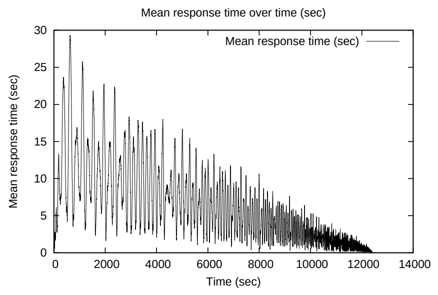
Table 7. Summary of experiments - Part 1

Id	Admission Control		Scheduling	Time	Max	Mean
	Session	Burst	Policy	(sec)	R (sec)	R (sec)
1	No	No	FIFO	12,604	29.41	6.51
2	React.	React.	FIFO	9,031	7.22	3.24
3	No	No	PFIN	13,002	84.07	6.27
4	No	No	IFPN	13,660	44.37	5.95
5	React.	React.	IFPN	8,910	7.12	3.35

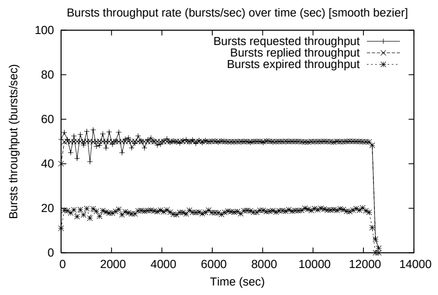
Table 8. Summary of experiments - Part 2

Id	Admission Control		Scheduling	Bursts	Bursts	Bursts	Burst	Rejected
	Session	Burst	Policy	requested	responded	expired	lost rate	sessions
1	No	No	FIFO	621,342	621,342	229,755	36.98%	0
2	React.	React.	FIFO	535,018	418,299	206,622	38.62%	717
3	No	No	PFIN	621,342	621,342	157,537	25.35%	0
4	No	No	IFPN	621,342	621,342	16,500	2.66%	0
5	React.	React.	IFPN	544,078	423,679	126,925	23.33%	640

In experiment 1 no admission control policy is active and the scheduling performed follows the typical FIFO approach. Figure 20 presents the average response time (a) and the bursts throughput (b). As we observe, the mean response time achieves a value higher than the 10-seconds limit due to the great number of requests being scheduled to the server. The throughput of the server is close to the server limit and there is a great number of bursts that expires during the execution, showing the overload situation. The total simulation time for the execution of the 5,000 sessions is 13,284 seconds. We observe that the burst lost rate achieves 36.98%, representing probably a high number of unsatisfied users.



(a) Average Response Time

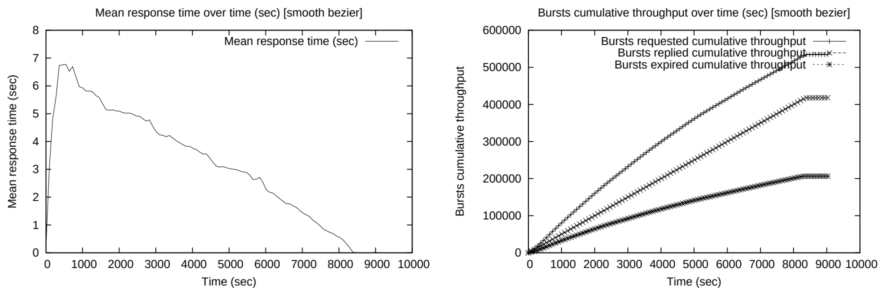


(b) Bursts Throughputs

Fig. 20. Experiment 1: no admission control and FIFO scheduling

The simulation performed in experiment 2 uses the reactive admission control policies for session and burst. The scheduling approach is set to run the typical FIFO approach. The response time achieves lower values than experiment 1, showing the experiment configuration is effective to guarantee a better QoS level. Table 7 shows the maximum response time of 7.22 seconds, almost the lower value compared to the other experiments. This is due to the rejection of sessions and bursts by the admission control mechanism. Figure 21 shows the response time achieved (a), the cumulative throughput (b), and the number of sessions (c). It is important to notice that there is a significant difference between the cumulative bursts requested throughput and the cumulative number of bursts replied, as we can see in (b).

Their difference corresponds to the number of bursts rejected by the reactive mechanism of admission control. Since a great number of bursts and sessions are rejected the load under the server decreases and the response time achieves lower values. Despite the low response times, the user satisfaction is not fulfilled and the burst lost rate achieves 38.62%, showing that a scheduling mechanism may improve the results.



(a) Average Response Time (b) Cumulative Throughput

Fig. 21. Experiment 2: Reactive admission control and FIFO scheduling

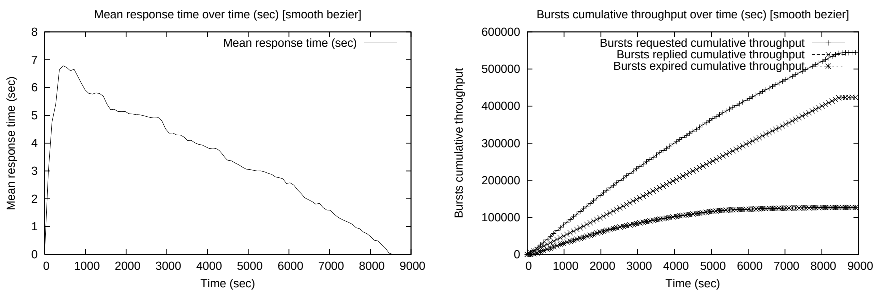
In experiment 3 and 4 there is no admission control active but the scheduling mechanism is set to perform the PFIN an IFPN approaches, respectively. For experiment 3 the busts lost rate achieves 25.35%, a lower value than experiments 1 and 2, but the response time achieves 84.07 seconds showing that the scheduling may cause high delays. For experiment 4, the response time achieves 44.37 seconds, but the lost rate is very low (2.55%) showing that the IFPN scheduling is very effective to provide a response time according to the user tolerance to QoS. Worths to remember that such mechanism gives higher priority to requests of impatient users. Those requests are answered first by the server and the impatience level decreases, impacting the burst lost rate.

In order to improve the results, experiment 5 evaluates the hybrid strategy. In this experiment the reactive mechanism of admission control for session and burst and the reactive IFPN scheduling approach are active. As we observe, the maximum response time values observed on Table 7 present the lower value compared to the other experiments. The experiment presents the second lower value for the burst lost rate (21.67%). Figure 22 presents the response time (a) and cumulative throughput (b). We observe the low response time values and a significant percentage of bursts rejected that impacts the burst lost rate value.

6.3.4 Discussion

The experiments evaluating the reactive strategies achieve better response times and burst lost rates compared to the experiment without any additional mechanism. The experiment 4, running just the IFPN reactive scheduling, achieves the best burst lost rate, otherwise the response time behavior is not the best one. Experiment 2 achieves better response times due to the effectiveness of the rejection of bursts and sessions, despite their high burst lost rate. Experiment 5, running the reactive admission control and scheduling, presents the best result, achieving an equilibrium in terms of response time and burst lost rate values.

Considering these results, we conclude that the reactive QoS approach present significant improvements, despite the gains are different according to each experiment configuration.



(a) Average Response Time (b) Cumulative Throughput

Fig. 22. Experiment 5: reactive admission control and IFPN scheduling

Moreover, it is a task of the systems engineer to choose the best approach in order to provide each application demands.

7 Conclusion

Several previous studies focused on characterization and generation of workloads from Internet service providers. However, as far as we know, none of them models the user reactions to the service response time. We believe that this information is very useful to understand Web workloads.

The basis of this work is the *USAR* characterization model, which guides the characterization of the user actions based on the latency and the IAT of the requests. The model also comprises a validation strategy, that is a starting point towards generating more realistic workloads. We validate the model through a case study, using the log of an actual proxy-cache server, where we demonstrate the key features of the *USAR* model.

The work also evaluates the impact of reactivity on the performance of Web applications. We design a new version of *httperf* workload generator that considers reactivity, based on *USAR* model [37]. Using this, we perform experiments, comparing the non-reactive and reactive approaches.

The results show that reactivity causes a significant impact on the server's performance. This can be explained by the static behavior assigned to clients in the non-reactive scenario. Adopting traditional workload generation mechanisms, the unavailability of the system is an expected situation, since changes in the users' reaction are not considered. Our new model shows the importance of understanding better the user-server interactivity process.

Moreover, this work presents novel contributions explaining how reactivity occurs, how it affects the system's performance, and how different user profiles reacts over variations on the server's performance. We design and implement the *USAR-QoS* simulator which allows the analysis of each user profile behavior.

Based on the reactivity, the work proposes new strategies to guarantee QoS in Internet services. We propose the use of admission control techniques based on the user reactivity which are capable of selecting requests according to an user-based criteria. We also propose the PFIN (patient-first impatient-next) and IFPN (impatient-first patient-next) scheduling approaches that schedules the incoming requests arriving on the server according to the classes of the reactivity model. In order to improve the benefits of admission control an scheduling we present a hybrid technique that combines both strategies in a single solution. The new strategies are evaluated using the *USAR-QoS* simulator and a TPC-W-based workload.

From the results we observe different gains according to each QoS approach. The mechanism based only on reactive scheduling achieves the best burst lost rate (i.e. the number of requests the user presents the impatient behavior), since its mechanism gives priority to requests classified with impatient classes. The mechanisms that adopt the admission control are effective to reduce the response time but may cause the increase in the burst lost rate, due to the increase in the amount of users rejecting bursts or sessions. The hybrid mechanism presents an equilibrium, reducing both the response time values and the burst lost rate.

Is important to observe that the proposed policies may be unfair to certain users since their typical behavior is used to improve QoS. The benefits obtained are significant and we understand that our policies may be very useful despite the subject of unfairness, specially in overload situations.

Finally, we observe that there is a relevant improvement in the QoS of reactive Internet systems through the use of reactive approaches. As part of ongoing work we plan to study the problem of *starvation* due to the scheduling, using some mechanism such as fair queuing [27]. We also intend to implement the strategies in a real web server.

References

1. Specweb99. `http://www.specbench.org/ osg/web99/` .
2. Tpc - transaction processing council. tpc benchmark w. `http://www.tpc.org/ tpcw/`.
3. Webbench. `http://www.veritest.com/ benchmarks/webbench/`.
4. T. F. Abdelzaher and N. Bhatti. Web content adaptation to improve server overload behavior. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1563–1577, 1999.
5. J. M. Almeida, J. Krueger, and M. K. Vernon. Characterization of user access to streaming media files. In *Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 340–341. ACM Press, 2001.
6. M. F. Arlitt and C. L. Williamson. Web server workload characterization: The search for invariants. In *Measurement and Modeling of Computer Systems*, pages 126–137, 1996.
7. M. F. Arlitt and C. L. Williamson. Internet web servers: workload characterization and performance implications. *IEEE/ACM Trans. Netw.*, 5(5):631–645, 1997.
8. A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan. Characterizing user behavior and network performance in a public wireless lan. *SIGM. Perf. Eval. Rev.*, 30(1):195–205, 2002.
9. P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 151–160. ACM Press, 1998.
10. N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating user-perceived quality into web server design. *Comput. Networks*, 33(1-6):1–16, 2000.
11. N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, 13(5):64–71, 1999.
12. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. Technical Report RFC 2475, IETF, 1998.
13. R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. Technical Report RFC 1633, IETF, 1994.
14. P. Chatterjee, D. Hoffman, and T. Novak. Modeling the clickstream: Implications for web-based advertising efforts, 1998.
15. H. Chen and P. Mohapatra. Overload control in qos-aware web servers. *Comput. Networks*, 42(1):119–133, 2003.

16. L. Cherkasova and P. Phaal. Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Trans. Comput.*, 51(6):669–685, 2002.
17. C. Costa, I. Cunha, A. Borges, C. Ramos, M. Rocha, J. Almeida, and B. R. Neto. Analyzing client interactivity in streaming media. In *Proc. of the 13th World Wide Web Conference*, 2004.
18. M. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. In *Proceedings of SIGMETRICS'96: The ACM International Conference on Measurement and Modeling of Computer Systems.*, Philadelphia, Pennsylvania, May 1996.
19. D. Feitelson. Workload modeling for performance evaluation, 2002.
20. T. Ferrari. End-to-end performance analysis with traffic aggregation. *Computer Networks*, 34(6):905–914, December 2000.
21. P. A. Fishwick. Simpack: Getting started with simulation programming in c and c++. In *Winter Simulation Conference*, pages 154–162, 1992.
22. D. F. Garca and J. Garca. Tpc-w e-commerce benchmark evaluation. *Computer*, 36(2):42–48, 2003.
23. S. Garner. Weka: The waikato environment for knowledge analysis. In *Proceedings of the New Zealand Computer Science Research Students Conference*, pages 57–64, 1995.
24. T. Henderson. Latency and user behaviour on a multiplayer game server. In *Proceedings of the Third International COST264 Workshop on Networked Group Communication*, pages 1–13. Springer-Verlag, 2001.
25. H. Hlavacs, E. Hotop, and G. Kotsis. Workload generation by modeling user behavior. In *Proceedings of OPNETWORKS 2000*, 2000.
26. H. Hlavacs and G. Kotsis. Modeling user behavior: A layered approach. In *MASCOTS*, pages 218–225, 1999.
27. M. Karlsson, C. Karamanolis, and J. Chase. Controllable fair queuing for meeting performance goals. *Perform. Eval.*, 62(1-4):278–294, 2005.
28. M. Kearns, Y. Mansour, and A. Y. Ng. An information-theoretic analysis of hard and soft assignment methods for clustering. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 282–293, Providence, Rhode Island, USA, 1997.
29. B. Krishnamurthy, Y. Zhang, C. Wills, and K. Vishwanath. Design, implementation, and evaluation of a client characterization driven web server. In *WWW'03: Proceedings of the twelfth international conference on World Wide Web*, pages 138–147. ACM Press, 2003.
30. D. Menascé, V. Almeida, and L. Dowdy. *Performance by Design.* Prentice Hall, 2004.
31. D. Menascé, V. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. M. Jr. A hierarchical and multiscale approach to analyze e-business workloads. *Perform. Eval.*, 54(1):33–57, 2003.
32. D. Menascé, V. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira, Jr. In search of invariants for e-business workloads. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 56–65. ACM Press, 2000.
33. D. A. Menascé. Testing e-commerce site scalability with tpc-w. In *Int. CMG Conference*, pages 457–466, 2001.
34. D. A. Menascé, V. A. F. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira, Jr. A hierarchical and multiscale approach to analyze e-business workloads. *Perform. Eval.*, 54(1):33–57, 2003.
35. D. Mosberger and T. Jin. httperf–tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37, 1998.
36. D. Olshefski, J. Nieh, and D. Agrawal. Inferring client response time at the web server. In *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 160–171. ACM Press, 2002.
37. A. Pereira, G. Franco, L. Silva, W. Meira, Jr., and W. Santos. The *usar* characterization model. In *Proceedings of the IEEE 7th Annual Workshop on Workload Characterization (WWC-7)*, Austin, Texas, USA, 2004. IEEE Computer Society.
38. A. Pereira, L. Silva, W. Meira, Jr., and W. Santos. Assessing reactive qos strategies for internet services. In *Proceedings of the IEEE/IPSJ Symposium on Applications and the Internet*

(SAINT2006), Phoenix, Arizona, USA, 2006. IEEE Computer Society (IEEE-CS) and IPSJ.

39. A. Pereira, L. Silva, W. Meira, Jr., and W. Santos. Assessing the impact of reactive workloads on the performance of web applications. In *Proc. of the IEEE Int. Symposium on Performance Analysis of Systems and Software (ISPASS-2006)*, Austin, Texas, USA, 2006. IEEE C. S.
40. P. Selvridge, B. Chaparro, and G. Bender. The world wide wait: Effects of delays on user performance. In *Proceedings of the IEA 2000/HFES 2000 Congress*, 2000.
41. E. Veloso, V. Almeida, W. Meira, A. Bestavros, and S. Jin. A hierarchical characterization of a live streaming media workload. In *Proceedings of the second ACM SIGCOMM Workshop on Internet measurment*, pages 117–130. ACM Press, 2002.
42. A. J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw.*, 3(3):253–256, 1977.
43. N. Ye, E. S. Gel, X. Li, T. Farley, and Y.-C. Lai. Web server qos models: applying scheduling rules from production planning. *Computers & Operations Research*, 32:1147–1164, 2005.