Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites

Ludmila Cherkasova and Peter Phaal

Abstract—In this paper, we consider a new, session-based workload for measuring web server performance. We define a session as a sequence of client's individual requests. Using a simulation model, we show that an overloaded web server can experience a severe loss of throughput measured as a number of completed sessions compared against the server throughput measured in requests per second. Moreover, statistical analysis of completed sessions reveals that the overloaded web server discriminates against longer sessions. For e-commerce retail sites, longer sessions are typically the ones that would result in purchases, so they are precisely the ones for which the companies want to guarantee completion. To improve web QoS for commercial web servers, we introduce a *session-based admission control (SBAC)* to prevent a web server from becoming overloaded and to ensure that longer sessions can be completed. We show that a web server augmented with the admission control mechanism is able to provide a fair guarantee of completion, for any accepted session, independent of a session length. This provides a predictable and controllable platform for web applications and is a critical requirement for any e-business. Additionally, we propose two new adaptive admission control strategies, hybrid and predictive, aiming to optimize the performance of SBAC mechanism. These new adaptive strategies are based on a self-tunable admission control function, which adjusts itself accordingly to variations in traffic loads.

Index Terms—Session-based web workload, overloaded web server, performance analysis, admission control, web QoS, adaptive control strategies, simulation, synthetic workload generator.

1 Introduction

As the Internet matures, companies are implementing mission critical Internet applications. These applications provide dynamic content, integrate with databases, and offer secure commercial transactions. Customers are becoming increasingly reliant on these complex business applications for services such as banking, product purchases, and stock trading. These new services make greater demands on web servers at a time when traffic is increasing rapidly, making it difficult to ensure an adequate level of service.

Evaluation of web server performance generally focuses on achievable throughput and latency for a request-based type of workload as a function of a traffic load. The SpecWeb96 benchmark [21], proposed few years ago as an industry standard for measuring a web server's performance, is based on generating HTTP requests to retrieve different length files accordingly to a particular distribution. The server performance (throughput) is characterized as a maximum achievable number of connections per second while maintaining the required file mix.

However, commercial applications impose a set of additional, service-level expectations. Typically, access to a web service occurs in the form of a *session* consisting of many individual requests. Placing an order through the web site involves further requests relating to selecting a product, providing shipping information, arranging payment agreement, and, finally, receiving a confirmation. So, for a customer trying to place an order or a retailer trying to make a sale, the real measure of a web server performance is its ability to process the entire sequence of requests needed to complete a transaction. In this paper, we introduce a new model of workload based on sessions. A session-based workload gives a new interesting angle to revisit and reevaluate the definition of web server performance. It naturally proposes to measure a server throughput as a number of successfully completed sessions.

Let us consider the situation when a server is processing a load that exceeds its capacity.

If a load consists of single, unrelated requests, then the server throughput is defined by its maximum capacity, i.e., a maximum number of connections the server can support. Any extra connections will be refused and extra load-requests will be dropped. Thus, once a server has reached its maximum throughput, it will stay there, at a server maximum capacity.

However, if the server runs a session-based workload, then a dropped request could occur anywhere in the session. That leads to aborted, incomplete sessions. Using a simulation model, we show that an overloaded web server can experience a severe loss of throughput when measured in completed sessions while still maintaining its throughput measured in requests per second. As an extreme, a web

- *L. Cherkasova is with Hewlett-Packard Laboratories, 1501 Page Mill Rd., Palo Alto, CA 94303. E-mail: cherkasova@hpl.hp.com.*
- *P. Phaal is with InMon Corp., 1404 Irving St., San Francisco, CA 94122. E-mail: Peter_Phaal@inmon.com.*

Manuscript received 24 Feb. 2001; revised 30 Nov. 2001; accepted 24 Jan. 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 115751.

0018-9340/02/$17.00 © 2002 IEEE

server which seems to be busily satisfying clients requests and working at the edge of its capacity could have wasted its resources on failed sessions and, in fact, not accomplished any useful work. Statistical analysis of completed sessions reveals that an overloaded web server discriminates against the longer sessions. Our analysis of a retail web site showed that sessions resulting in sales are typically 2-3 times longer than nonsale sessions. Hence, discriminating against the longer sessions could significantly impact sales and profitability of the commercial web sites.

The importance of understanding the session-based characterization of web server workloads has been recognized in several studies [8], [15], [2].

Quality of service is a way of describing the end-to-end performance requirements and conditions that a particular application imposes to be successfully executed. For a web server running a commercial application, the following *web quality of service* requirement is crucial:

- a fair chance of completion for any accepted session, independent of session length.

Peak load management is required to improve the customer perception of the web site quality of service and to ensure a quality end-user experience to win customer loyalty.

The main goal of a *session based admission control (SBAC)*, which we introduced in [8], is the prevention of a web server overload and, as a result, an improvement of the quality of service on a web site. An admission control mechanism will accept a new session only when a server has the capacity to process all future requests related to the session, i.e., a server can guarantee the successful session completion. If a server is functioning near its capacity, a new session will be rejected (or redirected to another server if one is available). The idea of introducing admission control for a web server running a session-based workload to guarantee web QoS closely resembles the admittance of a new flow into a resource constrained network in a traditional QoS setup [12], [4].

Deferring a client at the very beginning of their transaction (session)—rather than in the middle—is another desirable web quality of service property for an overloaded server. It will minimize the amount of wasted server work. We show that a web server augmented with a session-based admission control is able to provide a fair guarantee of completion, for any accepted session, independent of a session length. This provides a predictable and controllable platform for web applications and is a critical requirement for any e-business.

Hewlett-Packard's WebQoS product aims to improve the quality of service being offered by web servers during peak usage periods. This product deploys a proposed session-based admission control mechanism in order to ensure the high levels of service required to successfully complete commerce transactions on the web. A good survey describing the WebQoS architecture and implementation details is provided in [6]. WebQoS works on the premise that some visitors have greater priority than others. A customer who's completing a purchase transaction, for instance, is much more important than someone browsing for information. As such, WebQoS allows incoming requests to be screened and categorized as high, medium, or low priority based on things like source/destination IP address, requested URL, and so on. Different service-level policies can then be applied to each category. For example, if the server is experiencing heavy traffic, low priority requests can be redirected to an alternative server or receive rejection notices. WebQoS allows administrators to selectively manage Web site performance and availability, while, at the same time, providing extra protection against Denial of Service attacks.

We believe that sending a clear message of rejection to a client is very important. It will stop clients from unnecessary retries which could only worsen the situation and increase the load on the server. However, issuing an explicit rejection message imposes an additional load on a web server. In the paper, we present the "worst case" overhead analysis depending on workload type and applied load values.

There are two desirable, but somewhat contradictory, properties for an admission control mechanism: *stability* and *responsiveness*. It is well-known that web workloads exhibit bursty behavior. It is not unusual to observe a 120 percent load during a minute, followed by a 50 percent load during the next minute. In this case, when the server receives an occasional burst of new traffic, while still being under a manageable load, the stability, which takes into account some load history, is a desirable property for admission control mechanism. It helps to maximize server throughput and to avoid unnecessary rejection of newly arrived sessions. However, if a server's load during previous time intervals is consistently high and exceeds its capacity, the responsiveness is very important: The admission control policy should be switched on as soon as possible to control and reject newly arriving traffic. In this paper, we examine a trade-off between these two desirable properties for an admission control mechanism: *responsiveness* and *stability* and a family of admission control policies (ac-policies) which cover the space between ac-stable and ac-responsive policies.

Once a web server is augmented with an admission control mechanism, the following question arises: How do we measure the "goodness" and efficiency of this mechanism in practice? We use a novel quality of service metric based on aborted connections to measure the effectiveness of the admission control mechanism.

Additionally, we consider a new *hybrid* admission control strategy, which tunes itself to be "more responsive" or "more stable" on the basis of observed quality of service. We show that the proposed hybrid strategy successfully combines the most attractive features of both ac-responsive and ac-stable policies. It demonstrates improved performance results for workloads with medium to long average session length.

We analyze why workloads with short average session lengths are the most difficult to manage. We design a new, *predictive* admission control strategy, which estimates the number of new sessions a server can accept and still guarantee processing of all future session requests. This adaptive strategy evaluates the observed workload and makes "its prediction" for the load in the nearest future. It consistently shows the best performance

results for different workloads and different traffic patterns. For workloads with short average session length, the predictive strategy is the only strategy which provides both: highest server throughput in completed sessions and no (or practically no) aborted sessions.

The proposed hybrid and predictive admission policies allow the design of a powerful admission control mechanism which tunes and adjusts itself for better performance across different workload types and different traffic loads.

2 Related Work

A few mechanisms have been proposed to support quality of service and admission control mechanism in web servers [1], [19], [14], [3], [13], [16], [7].

In [1], the quality of service is introduced by assigning the priorities to requests based on the requested content. The metric for quality of service is latency in handling the HTTP requests. The priority mechanism is static and limited to priority schemes for CPU scheduling. In [19], the authors develop a quality of service model that implements algorithms for scheduling CPU, memory, and networking resources. In their model, a site can determine how requests for various pages can be served. Both of the papers above do not propose an active admission control mechanism for server resource management. In [14], the authors develop a measurement-based admission-controlled web server which is able to allocate a configurable fixed percentage of server bandwidth across numerous simultaneous client requests. Their algorithm will reject requests from the client when it has both received more than its allocation of the bandwidth and the server if fully utilized. The proposed mechanisms in [1], [19], [14] are explicitly oriented on request-based workload model, which is quite different from the commercial sites' session-based workload model proposed in our paper. Thus, the admission control mechanism introduced in [14] will drop any session in progress if this session exceeds the assigned percentage of server bandwidth utilized by the session. However, "fair" bandwidth allocation is not the best policy to apply for commercial web servers. Since the session can be aborted any time while in progress, this could lead to inefficient system resource utilization. Aborted sessions can be considered to not be performing useful work while "wasting" system resources. Moreover, such a strategy could result in unsatisfactory user experience at those sites.

Today's web servers often perform poorly under overload. Several techniques have been proposed to alleviate server overload, such as methods for distributing the load across a cluster of geographically replicated servers [10], [11]. In order to meet quality of service requirements, system administrators are forced to significantly overprovision their sites. These solutions are based on load balancing among the multiple servers. In the e-commerce environment, the concept of *session* plays an essential role. HTTP protocol is stateless, i.e., each request is processed by the web server independently from previous or subsequent requests. For a session, it is important to maintain state information from the previous interactions between a client and a server. Such a state might contain the content of the shopping cart or a list of results from the search request. For efficient request processing and *session integrity*, it is desirable to send the client request to the same server. One of the popular schemes proposed for handling the state on the web is cookies. Content-aware request routing [18], [9], [22] provides a convenient mechanism to support session integrity (the other common term for this is "sticky" connection). Some other load balancing solutions provide "sticky" connections by sending the requests from the same IP address to the same server. Session-based admission control, designed in our paper, can be easily adopted for web server cluster configuration, where a load balancing solution supports a "sticky" connection concept.

Some alternative solutions to this problem were proposed in [3], where the server overload is alleviated by providing the clients with a degraded, less resource intensive version of requested content. Adaptation software described in this paper implements a mechanism for measuring server load that can be used to toggle between two modes: to use high-quality delivered content and a degraded content mode. Client-identifier hashing can be used to pick the subset of clients who will be served with degraded content. While this technique can be very useful when the server overload is not very high, it cannot guarantee quality of service for all the arriving sessions under high overload conditions. If the number of arriving client sessions is higher than server capacity when it operates in degraded content mode, then it will inevitably lead to aborted and rejected sessions. Thus, the proposed technique can benefit when combined with an active session-based admission control mechanism proposed in our paper.

In [13], the authors propose a new framework for multiclass web server control which can satisfy per-class latency constraints. Their algorithm uses measurements for requests and service latencies to control each class's quality-of-service. The goal of their admission control algorithm is to determine whether admission of a new request in a particular service class can be supported while meeting the latency targets of all classes. The paper presents an elegant abstraction of system resources into a high-level virtual server, avoiding modeling of the complex interactions of low-level system resources. However, the queuing model designed in the paper is applicable to request-based workloads and, additionally, one of the model assumptions is that the admission control decision takes negligible time (i.e., the actual processing time spent while making the admission control decisions is ignored). Thus, it is not clear, how this model can be extended for session-based workloads with explicit rejection-response messages, which are essential for commercial sites during the overload periods.

In [16], the authors successfully argue that resource management for e-commerce sites should be geared toward optimizing business metrics as opposed to conventional performance metrics. They present a priority-based set of policies where priorities are changed dynamically as a function of customer state and the amount of money the customer accumulated in his/her shopping cart. They use three priority classes: high, medium, and low. All new sessions start as a high priority class. After the session

TABLE 1
Session Length Distribution for e-Commerce Workload

	Mean Session Length	Percentage of Sessions
Total	36.5	100%
Sales	73.6	18.3%
Non-Sales	28.2	81.7%

length exceeds the threshold *m1* and the customer has not added anything in the shopping cart, its priority is lowered to medium class, etc. This work proposes an interesting "business metrics" rationale for their policies. However, their model requires detailed knowledge about the main set of user behavior patterns at a visited web site. The parameters of designed policies heavily depend on this knowledge. It is highly desirable to propose an automatic way to extract these parameters, especially taking into account the dynamic and evolutionary nature of web site workloads. Since the authors do not introduce an active admission control mechanism, the proposed dynamic-priorities cannot help avoid abortion of sessions in progress when the server gets overloaded.

Paper [7] can be considered as a direct extension to the technique proposed in our paper. It describes a Web2K server augmented with QoS capabilities. Based on operator-specified prioritization criteria, the server classifies incoming requests as belonging either to the *premium* class or the *basic* class. Instead of passing incoming requests to the web server in a first-come-first-serve manner, the requests from the premium class are served first. When the server detects the overload, the Web2K server performs an admission control to avoid overcommitting its resources, similarly to the approach proposed in our paper, but additionally giving priority of acceptance to the premium class sessions.

3 Sessions Length Distribution for Commercial Web Sites

In order to outline a workload space of interest and narrow the simulation space, we have analyzed web server access logs from a particular commercial site. This commercial site allows small businesses to purchase products online. This site provides the clients with product catalogues to browse, the ability to add selected products to a "shopping cart," and, finally, to purchase the contents of the shopping cart, completing the sale. The size of the accessed web pages ranged from 200 bytes to 100 Kbytes. We analyzed a session length distribution, specific for sales and nonsales transactions. The distributions clearly show that the sale sessions are much longer than nonsale sessions. Table 1 summarizes the distribution statistics.

The average session length of *a sale* is more than 2.5 times longer that of a *nonsale*. For the presented workload, it is very important that the long sessions corresponding to sale transactions have a fair chance of completion under server overload conditions. Failure to provide such guarantees could significantly impact sales and profitability of the site.

The workload of a popular online banking web site is described in [7]. Each user session includes such steps as user logging in to the web site, checking the balance in his/her account, browsing through the last transactions posted to one of his/her accounts, transferring money between the accounts, and, finally, logging out. Each session included about 40 unique URLs, with the size ranging from 200 bytes to 20 Kbytes. Average session is lasting a few minutes with the total user think time during the session being about 75 sec.

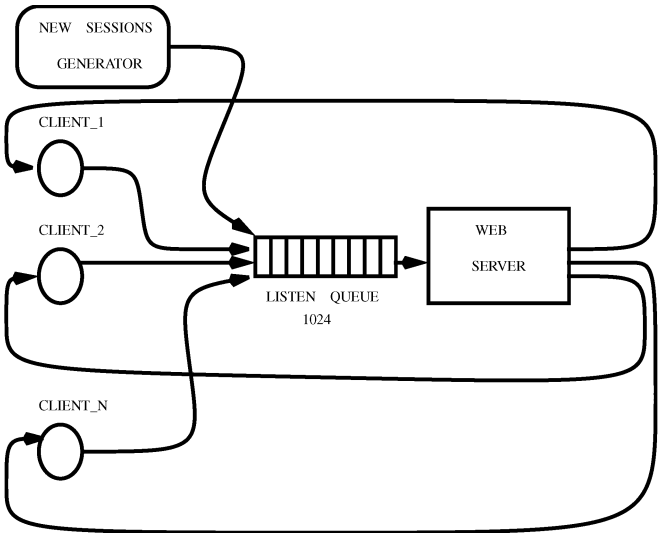

Fig. 1. The basic structure of the simulation model.

While usage of real traces for simulation purposes has advantages: 1) The trace-based approach is relatively easy to implement and 2) real traces imitate the activities of known systems in a straightforward way—the trace-based approach has the main drawback that it treats a workload as a "black box." Logical insight into the causes of system behavior is hard to obtain. Furthermore, it is hard to adjust these workloads to imitate conditions of varying load demand [5], [17]. The real traces reflect the workload which the server has already accepted for processing; they do not reflect dropped or rejected client requests/connections which are typical under high load conditions.

Thus, for simulation purposes, we decided to use the synthetic workload generator rather than the real traces from commercial sites because it allows us to perform the sensitivity analysis in a flexible way. By varying the parameters in the workload generator, we can analyze and predict the behavior of the *SBAC* mechanism across the different range of workloads, derive specific properties of the proposed mechanism, as well as identify its potential problems.

4 Simulation Model for Session-Based Workload

In order to understand the difference in web server performance when it runs request-based versus session-based workloads and to compare different admission control strategies, we built a simulation model using C++Sim [20]. The basic structure of the model is outlined in Fig. 1. It consists of:

- a session workload generator,

- N clients (we had limited N to 10,000 clients),
- a web server.

We introduce the notion of a session as a unit of session workload. *Session* is a sequence of clients' individual requests. A session workload generator produces a new session request according to specified input model parameters:

- session load and
- sessions length distribution.

In our simulation, the session structure is defined by the client (sender) address, original session length, current session length, and the timestamp when the session was initiated. For a new session, the original and current session lengths coincide. For a session in progress, the current session length reflects the number of requests left to complete.

We define a request as a structure specified by the following parameters:

- the session that originated the request,
- requested file size,
- time stamp when the request was issued.

Throughout this paper, we consider a file mix as defined by SpecWeb96 [21]. So, the individual requests retrieve the files defined by the following four classes:

- *Class 0:* 100 bytes-900 bytes (35 percent),
- *Class 1:* 1 Kbytes-9 Kbytes (50 percent),
- *Class 2:* 10 Kbytes-90 Kbytes (14 percent),
- *Class 3:* 100 Kbytes-900 Kbytes (1 percent).

For the rest of the paper, we assume a server capacity is 1,000 connections per second for a SpecWeb96-like file mix and that the service time for a file (request) is proportional to the requested file size.[1] We intentionally eliminated network delay from our model in order to concentrate on the effect of an overloaded web server.

A session request (i.e., the first request of a session) is sent to a web server and is stored in the server *listen queue*. We limit the size of the listen queue to 1,024 entries, which is a typical default value.

In this way, we are able to use an open model for sessions generation. Each consequent request from a session is issued and handled by a specified client. Client behavior is defined by a closed (feedback) loop model: The client issues the next session request only when it receives a reply from the previous request. The client issues its next request with some time delay, called *think time*. Think time is a part of the client definition rather than a session structure. The client waits for a reply for a certain time, called *timeout*. After a timeout, the client may decide to repeat its request—this is termed a retry. A limit is placed on retries—if this limit is reached and the reply is not received in time, both the request and the whole session are aborted.

Thus, a client model is defined by the following parameters:

- client address,
- think time between the requests of the same session,
- timeout—a time interval where the client waits for a server reply before reissuing the request,
- the number of retries before the session is aborted.

A session is successfully completed when all its requests are successfully completed. We will evaluate web server performance in terms of successfully completed sessions.

Two reasons could cause a request, and the session it belongs to, to be aborted:

- If a listen queue is full, then the connection to a server is refused and both the request and the whole session is aborted.
- After issuing the request, the client waits for a server reply for a certain time. After timeout, the client resends the request. There are a limited number of *retries*. If the reply still has not been received in time, both the request and a whole session are aborted.

Remark. Typically, when the client browser receives a "connection refused" message due to a full listen queue, it will try to resend the request again. In the case of an overloaded server, it only can worsen the situation. We decided to simplify the model by aborting the request and the whole session when a listen queue is full, without an additional client retry. Analysis provided in the Section 5 shows the model sensitivity to a number of client retries in more detail.

5 Analysis of Client Parameters: Think Time, Timeout, Number of Retries

Since we would like to evaluate a web server performance in terms of successfully completed sessions—the session length distribution is one of the main parameters in our sensitivity study of overloaded web server performance. As for the other client parameters such as timeout, the number of retries, and think time, we have analyzed their impact to narrow the simulation space.

There is a simple relation based on

- the server capacity,
- listen queue size, and
- the client timeout value

which defines the probability of retries issued by the client in our simulation model. Since a server processes 1,000 connections per second and the listen queue length is 1,024, the latency to process any accepted request is less than 2 seconds. If the client timeout is greater than 1 sec., then it eliminates the possibility of client timeouts and retries in the model. Since we are interested in studying a model with a full range of possible client-server interactions, we selected a timeout of 1 sec. A client timeout of 1 sec. might be considered as an additional quality of service requirement: It sets a limit on a request latency (server side latency)

1. In order to demonstrate the difference in web server performance for processing request-based versus session-based workloads, we do not need a response-file distribution that perfectly reflects today's usage patterns. It is merely sufficient that the workload have a reasonably high variance of response file size, as do today's usage patterns. Dynamic pages used for the subset of content at the commercial sites typically are much more resource-intensive (CPU-consuming) for a server to perform. Since the service time in our model is proportional to the requested file size, we can use this distribution to represent the mix of static and dynamic pages with high variance in the server CPU time needed to prepare those responses.

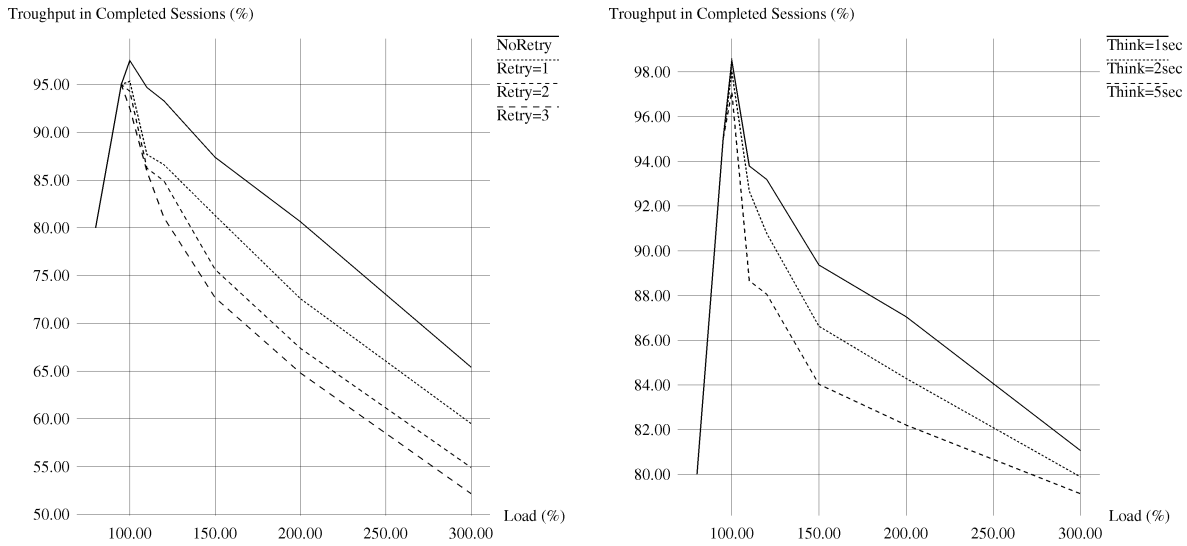

Fig. 2. Model sensitivity to different client parameters: (a) number of retries, (b) client think time. Workload in both examples is defined by the average session length of 15.

of 1 sec. If this requirement is not met (after a given number of retries), the session is aborted.

To pick some values for the number of client retries, we ran a simulation model with session lengths to be exponentially distributed with a mean of 5, 15, and 50, correspondingly, and a number of client retries varying between 0 and 3. Fig. 2a shows a sensitivity of web server throughput in completed sessions to the number of client retries (for a workload with average session length of 15). As one could expect, the higher number of retries leads to a worse server throughput in completed sessions. The results for workloads with average session length of 5 and 50 are similar. To narrow the simulation space and to simplify the future analysis, we used a client model with one retry for the rest of the paper.

Fig. 2b shows a sensitivity of web server throughput in completed sessions to a client think time, where a think time between the requests of the same session is exponentially distributed with a given mean: 1 sec., 2 sec., and 5 sec., correspondingly. Our choice of exponential distribution was motivated by two different types of think times between the client requests (we used terminology proposed in [5]):

- *Active* think time, which separates different requests for the same page objects. Typically, it is short, less than 1 sec. [5].
- *Inactive* think time, which separates client requests for different web pages. Typically, this think time is longer: A client requires some time to view the received page before issuing the next page request.

The higher client think time leads to a slightly worse server throughput in completed sessions. It can be explained by the fact that, under higher client think time, there are a higher number of sessions which are currently in progress (session "life" duration is longer). This potentially leads to higher burstiness and chances of collisions for server resources among the sessions in progress. Since we are interested in recreating these bursty conditions, we have chosen a timeout value of 5 sec. for our experiments in the paper.

Thus, for the rest of the paper, we use a simulation model with the following server and client parameters:

- A server capacity is 1,000 connections per second for SpecWeb96 file mix,
- The service time for a file (request) is proportional to the requested file size,
- A timeout—the time client waits for a reply before resending the request—is set to 1 sec.,
- Think time between the requests of the same session is exponentially distributed with a mean of 5 sec.,
- The number of retries to resend the request after timeout is 1.

6 Characteristics of an Overloaded Web Server

In order to analyze the server performance depending on a session length, we have performed experiments for three workloads with session lengths being exponentially distributed with a mean of 5, 15, and 50.

Fig. 3a shows throughput in completed sessions for an overloaded web server.

At first glance, the server throughput in completed sessions looks somewhat acceptable. However, the ordering is somewhat counterintuitive: Web server performance is better for workloads with a longer session mean.

How can it be explained?

First of all, the server throughput is measured as the number of completed sessions. We have sessions of different length since the session lengths are exponentially distributed. Our first explanation of the above phenomenon is that shorter sessions have a higher chance of completion.

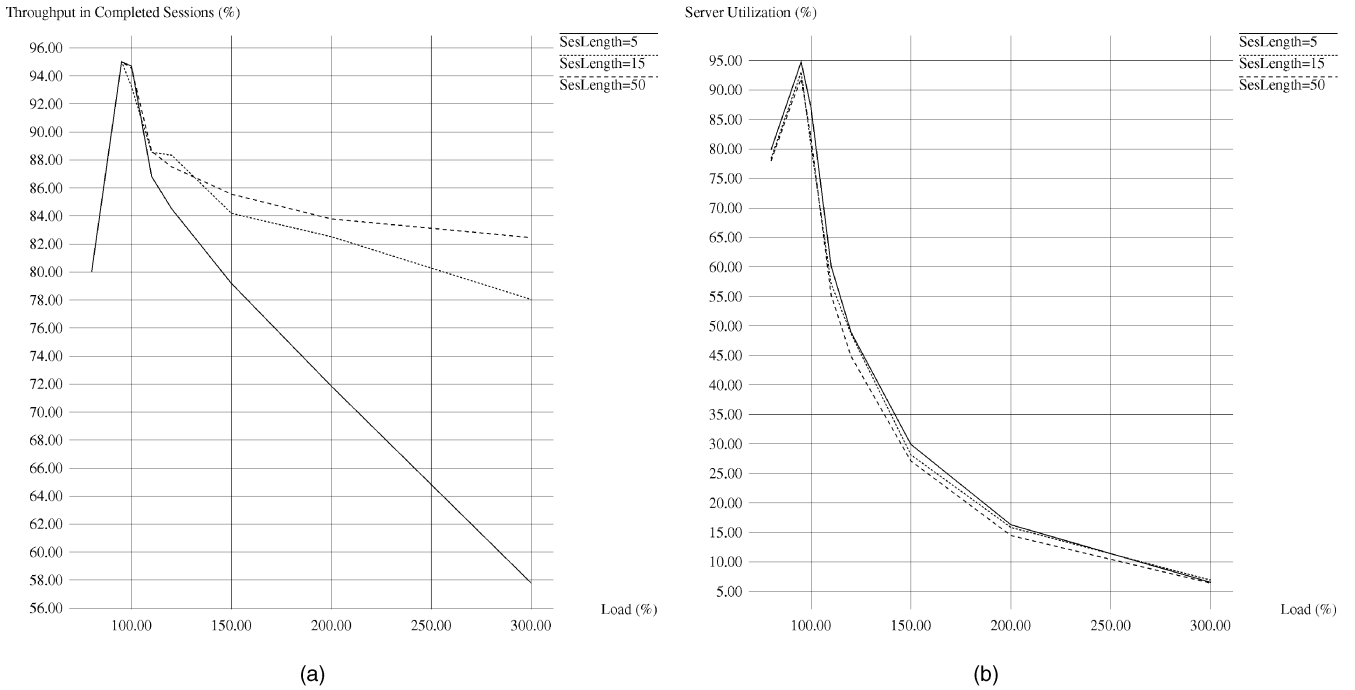

Fig. 3. (a) Throughput in completed sessions for an overloaded server. (b) Server useful utilization of processing sessions which complete.

Thus, the acceptable "quantitative" value of throughput can be obtained at a price of "lower quality" of this throughput.

The second explanation is of a different nature. The unit of our workload is a session represented by a sequence of clients individual requests. Let us consider a session length distribution with a mean of 50. When a long session gets aborted, it creates a potential amount of unused server resources (due to the "not send" sequence of client requests) big enough to service several short sessions. Applying the same reasoning to a session length distribution with a mean of 5, we can see that the difference between "long" and "short" session lengths for this distribution is less significant: Often, several sessions are aborted before it creates enough "unused" server resources to complete an additional session.

Let us analyze the simulation results in detail. Fig. 4a shows the average session length of completed sessions against the average session length of all generated sessions as the model input. As is clearly seen, the average session length of completed sessions is significantly lower than the original, input distribution. For the load of 300 percent and the original average session length of 5, 15, and 50, the average length of completed sessions is 1.7, 4.3, and 13.4, correspondingly.

The session lengths are defined to be exponentially distributed with a given mean m. In order to analyze the distribution of the completed sessions in more detail, we partition them in the following three bins: the first bin—the sessions shorter than or equal to m; the second bin—the sessions longer than m but shorter than or equal to $2 * m$; the third bin—the sessions longer than $2 * m$.

Fig. 4b shows the percentage of original and completed sessions in three bins by length for an overloaded server running a session-based workload with a mean of 50. The original distribution by session lengths is the following: the first bin—63 percent; the second bin—23 percent; and the third bin—14 percent. The distribution of completed sessions under 300 percent load changes dramatically: the first bin—98.14 percent; the second bin—1.83 percent; and the third bin—0.03 percent.

Indeed, the overloaded web server discriminates against the medium and long sessions in a quite severe way: Almost all the completed sessions fall in the first bin; the sessions from the second and the third bins are practically absent.

To complete the analysis of an overloaded web server running a session-based workload, we introduce a new performance measure: *useful server utilization*. Traditionally, a server performance is characterized by its throughput and utilization. We have shown a difference in the throughput of an overloaded web server when measured in percentage of completed requests and in percentage of completed sessions. We apply the same idea to characterize server utilization. We define *useful server utilization* as server busy time spent processing only sessions which complete. Fig. 3b shows useful server utilization as a function of load and session length. The results are overwhelming: The overloaded, "busy looking" server produces an amazingly small amount of useful work: around 15 percent for a 200 percent load; less than 7 percent for a 300 percent load.

This concludes our preliminary performance analysis of an overloaded web server characterization running a session-based workload. This section raises a rather serious question: Is such server behavior expected and acceptable for commercial sites? Since the answer is rather obvious, the next question to ask is: Can a web server be augmented with a session-based admission control mechanism to prevent the server from becoming overloaded and to ensure that longer sessions are completed?

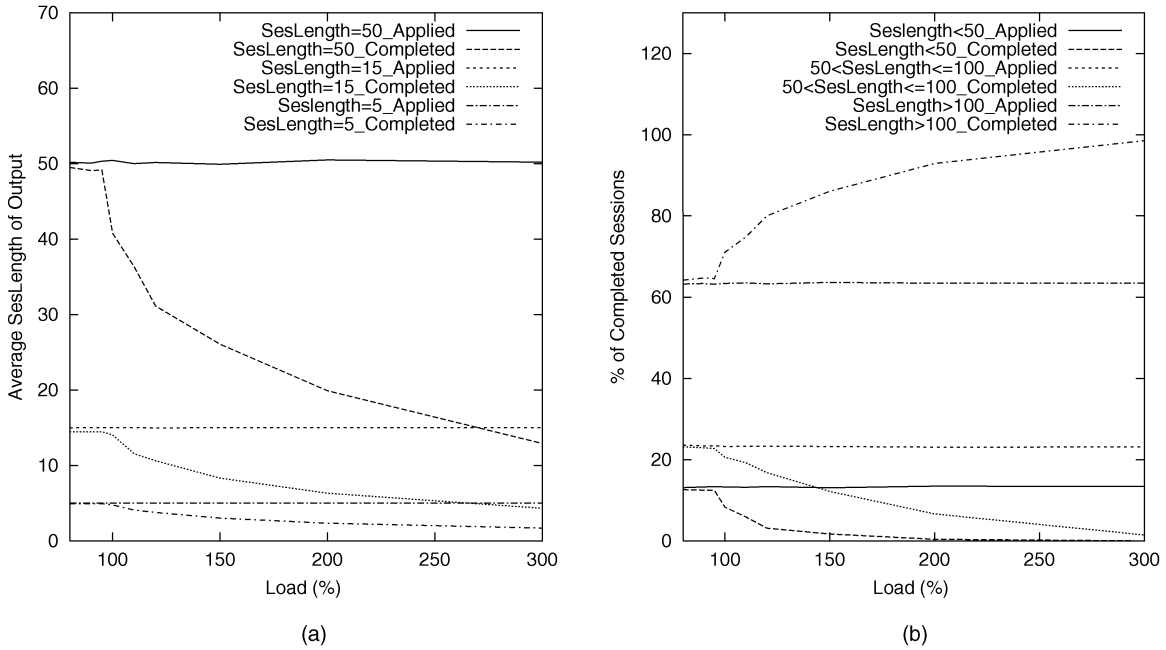

Fig. 4. Overloaded server running session-based workload: (a) Average length of completed sessions for workloads with average session length of 5, 15, and 50. (b) Percentage of completed wessions in three bins for average session length of 50.

7 "Utilization-Based" Implementation of the SBAC Mechanism

We use the term *web quality of service* to describe the service-levels needed to complete web sessions. A web server that ensures a fair opportunity and guarantee of completion for all sessions, independent of session length, exhibits good web quality of service. To satisfy the web quality of service requirements, we introduce a *session-based admission control mechanism* for a server handling a session-based workload. The main goal of an admission control mechanism is to prevent a web server from becoming overloaded. We introduce a simple admission control mechanism based on the server CPU utilization.[2]

The basic idea of a session-based admission mechanism is as follows: The server utilization is measured during predefined time intervals (say, each second). Using this measured utilization (for the last interval) and some data characterizing server utilization in the recent past, it computes a "predicted" utilization. If the predicted utilization gets above a specified threshold, then, for the next time interval, the admission controller will reject all new sessions and will only serve the requests from already admitted sessions.[3] Once the predicted utilization drops below the given threshold, the server changes its policy for the next time interval and begins to admit new sessions again.

2. In this work, we assume that the commercial web server is CPU bounded. If the web server has a different resource that is the major bottleneck (i.e., i/o or network), then the admission control mechanism is defined with respect to the utilization of the corresponding resource in a similar way.

3. The HP WebQoS product, which deploys the SBAC mechanism proposed in the paper, augments each client session with a cookie, which allows it to distinguish between newly arrived sessions and sessions in progress. The *expire* field in the cookie is used to define a time threshold T for client session termination: If client does not send any requests during T time period, the session is terminated.

Formally, the admission control mechanism can be defined by the following parameters:

- U_{ac}—an *ac-threshold* which establishes the critical server utilization level to "switch on" the admission control policy;
- $T_1, T_2, \ldots, T_i, \ldots$—a sequence of time intervals used for making a decision whether to admit (or to reject) new sessions during the next time interval. This sequence is defined by the *ac-interval length*;
- f_{ac}—an *ac-function* used to evaluate the predicted utilization.

We distinguish two different values for server utilization:

- $U_i^{measured}$—a measured server utilization during T_i—the ith ac-interval;
- $U_{i+1}^{predicted}$—a predicted utilization computed using a given ac-function f_{ac} after ac-interval T_i and before a new ac-interval T_{i+1} begins, i.e., $U_{i+1}^{predicted} = f_{ac}(i+1)$.

In this paper, we consider ac-function $f_{ac}(i+1)$ defining $U_{i+1}^{predicted}$ in the following way:

- $f_{ac}(1) = U_{ac}$;
- $f_{ac}(i+1) = (1-k) * f_{ac}(i) + k * U_i^{measured}$, where k is a damping coefficient between 0 and 1, and it is called an *ac-weight coefficient*.

A web server with an admission control mechanism reevaluates its admission strategy on intervals $T_1, T_2, \ldots, T_i, \ldots$ boundaries. Web server behavior for the next time interval T_{i+1} is defined in the following way:

- If $U_{i+1}^{predicted} > U_{ac}$, then any new session which arrived during T_{i+1} will be rejected and the web server will process only requests belonging to already accepted sessions.

- If $U_{i+1}^{predicted} \leq U_{ac}$, then the web server during T_{i+1} is functioning in a usual mode: processing requests from both new and already accepted sessions.

Session-based admission control combines several functions: 1) It measures and predicts the server utilization, 2) it rejects new sessions when the server becomes critically loaded, and 3) it sends an explicit message of rejection to the client of a rejected session. In our simulation model, we assume that processing a session rejection is equivalent to processing an average size file because we send an explicit rejection message.

The admission control mechanism should combine two desirable properties: *responsiveness* and *stability*. If a server's load during previous time intervals is consistently high and exceeds its capacity, then "fast reaction" responsiveness is very important: The admission control policy (AC-mechanism) should be "switched on" as soon as possible to control and reject newly arriving traffic. However, if the server receives an occasional burst of new traffic while still being under a manageable load (which is a very typical traffic pattern for web workloads), then the "slow reaction" stable admission control policy, which takes into account some load history, is a desirable property. It will help to maximize server throughput and will not unnecessarily reject newly arriving traffic. These two properties are somewhat contradictory:

- Responsiveness leads to a more restrictive admission policy. It starts to reject new sessions at the first sign of high load. It minimizes the number of aborted sessions, but achieves higher levels of service at a price of lower server session throughput (in particular, when a server operates under high load but is not yet overloaded).
- Stability takes into account a server's load history. It allows delay of the first reaction of an admission control policy to the overload. If a total server load is still around the server capacity, then such a strategy allows better server session throughput to be achieved. However, a less restrictive rejection policy inevitably leads to a higher rate of aborted sessions under server overload and, as result, to poorer session completion characteristics.

The value of coefficient k in the definition of f_{ac} introduces a family of admission control policies which cover the space between ac-stable and ac-responsive policies. In other words, the different values of coefficient k put a different "weight" on the importance and impact of the load history while computing the predicted server utilization for the next ac-interval. If $k = 1$, then the admission control policy is based entirely on the value of measured server utilization during the last ac-interval. Let us call this strategy *ac-responsive*. If $k = 0.1$, then the admission control policy decision is strongly influenced by a server load prehistory, while the impact of a measured server utilization during the last ac-interval is limited.[4] Let us call this strategy *ac-stable*.

8 Cost of Rejection

We believe that sending a clear message of rejection to a client is very important. It will stop clients from unnecessary retries which could only worsen the situation and increase the load on the server. If the server promises to serve these clients, say in five minutes, it might be enough to resolve the current overload and provide a high level of service without losing customers. Commercial sites might use some additional stimuli and bonuses issued in these rejection messages to keep their customers satisfied. However, issuing an explicit rejection message imposes an additional load on a web server. The higher the load—the greater the number of rejection messages sent by the server. How large is the rejection overhead? What percentage of total messages constitutes the rejection messages?

This section derives a worst case bound to estimate the rejection overhead as a function of the applied load and average session length. We use the following denotations:

- S_r—a server capacity in requests, i.e., the number of connections (requests) per second a server can sustain.
- S_s—a server capacity in sessions, i.e., the number of sessions per second a server can complete.
- $SesLength$—an average session length.
- $Load$—an applied load in sessions ($Load = 2$ means a load of 200 percent of server capacity).
- x—the number of rejected sessions per second.
- y—the number of completed sessions per second.

First of all, there is a simple relation between S_r, S_s, and $SesLength$:

$$S_s = \frac{S_r}{SesLength}. \tag{1}$$

Since S_s is a server capacity in sessions and $Load$ is an applied load in sessions, $Load * S_s$ is the total number of issued sessions per second. Obviously, the sum of the completed and rejected sessions per second is the number of sessions, in total, a server has received per second:

$$x + y = Load * S_s. \tag{2}$$

There are two types of sessions: completed and rejected ones. Each completed session implies that a client consequently makes, on average, the number of requests defined by the $SesLength$. Each rejected session is equivalent to

4. One can design a different family of admission control strategies which computes $U_{i+1}^{predicted}$ as an average server utilization measured during the n last ac-intervals. Under $n = 1$, this would lead to a similar ac-responsive strategy as proposed above because it is based only on a server utilization measured during the last ac-interval. However, under such an approach, it will be more difficult to design a strategy where the measurements of the last interval impact the predicted (computed) server utilization at, say, 70 percent. In order to do that, one still needs to introduce some ac-weight coefficient similar to the one proposed above when computing $U_{i+1}^{predicted}$ as a weighed average server utilization using the n last ac-interval of the utilization history.

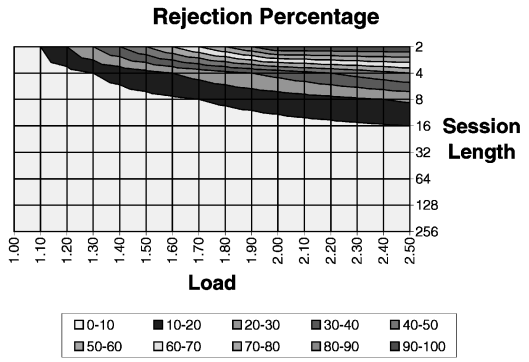

Fig. 5. Rejection cost as a percentage of rejection messages to a total number of requests per second.

processing a single request—a worst case estimate of the cost of sending an explicit rejection message to the client. Thus, the number of requests per second handled by a server is defined in the following way:

$$y * SesLength + x = S_r. \tag{3}$$

Replacing S_s in (2) with (1) and expressing y from (2), we have the following equation:

$$y = \frac{Load * S_r}{SesLength} - x. \tag{4}$$

Replacing y with (4) in (3), we can express x:

$$x = \frac{S_r * (Load - 1)}{SesLength - 1}. \tag{5}$$

Since x is the number of rejected sessions (rejection messages) per second and S_r defines the total number of requests per second processed by a server, then the percentage of rejection messages from the total number of requests is defined as follows:

$$\frac{100\% * x}{S_r}.$$

Let us call this percentage the $RejectionOverhead$. Here is the final equation:

$$RejectionOverhead = 100\% * \frac{(Load - 1)}{SesLength - 1}. \tag{6}$$

The rejection overhead depends on the average session length and the load received by the server.

Remark. Formula (6) holds for the $Load$ and $SesLength$ values, satisfying the following condition: $Load - 1 \leq SesLength - 1$. For the other values, (6) is meaningless and reflects the situation that the applied load is so high that the server's capacity is not enough to send all the rejection messages.

Fig. 5 illustrates the rejection overhead as the percentage of rejection messages to the total number of requests per second. Areas of different colors define corresponding rejection overhead (across different values for session length and load). As we can see, in this figure, the light area corresponds to a rejection overhead between 0-10 percent. In other words, for a workload with an average session length greater than 15 and a load up to 250 percent, the rejection overhead is less than 10 percent.

Clearly, the rejection cost varies depending on the average session length and applied load: the higher the load and the shorter the session length, the higher the rejection overhead. However, for most of the load values and workloads of interest, the overhead is less than 10 percent.

9 Characteristics of an Overloaded Web Server with Session-Based Admission Control

This section analyzes the simulation results of an overloaded web server augmented with session-based admission control. We analyze the results produced by the ac-responsive admission control policy introduced in Section 7 (i.e., ac-weight coefficient $k = 1$) with the following parameters:

- ac-threshold $U_{ac} = 95\%$,
- ac-interval length of 1 second.

A web server augmented with such an admission control policy reevaluates its admission strategy each second. Since the ac-responsive policy, $U_{i+1}^{observed}$ is defined entirely by the cpu utilization measured during the ith second, i.e., $U_{i+1}^{observed} = U_i^{measured}$.

If measured cpu utilization for the previous ith second is above the ac-threshold, i.e., $U_i^{measured} > 95\%$, then any new session arriving during the next second will be rejected and the web server will process only requests belonging to already accepted sessions. Otherwise, for the next second, the web server is functioning in its usual mode: processing requests from both new and already accepted sessions.

We performed the experiments for the average session lengths of 5, 15, and 50. We varied the load from 80 percent to 300 percent. The session workload with a mean of 5 is not a realistic representative of commercial workloads. However, we included this case to cover the simulation space and understand the possible admission control limitations. The same can be said about a load of 300 percent: If a web server is consistently overloaded more than 200 percent, it is time to increase the site capacity and to extend it with an additional server. However, for completeness and to understand the general behavior of the ac-mechanism, we included a load of 300 percent, too.

Fig. 6a shows throughput in completed sessions. At first glance, the only results for sessions with a mean length of 50 look perfect. In order to explain the curves corresponding to sessions with mean of 15 and 5, we need to remind one of the important details related to sessions rejection. Under high load, more sessions are rejected and the rejection overhead increases correspondingly. The percentage of completed sessions is largely offset by that amount.

One of the goals of the admission control mechanism is to minimize the number of aborted sessions (ideally, reducing them to 0) by explicit session rejection. Fig. 6b shows the percentage of aborted sessions from those admitted for processing. The results for sessions with a

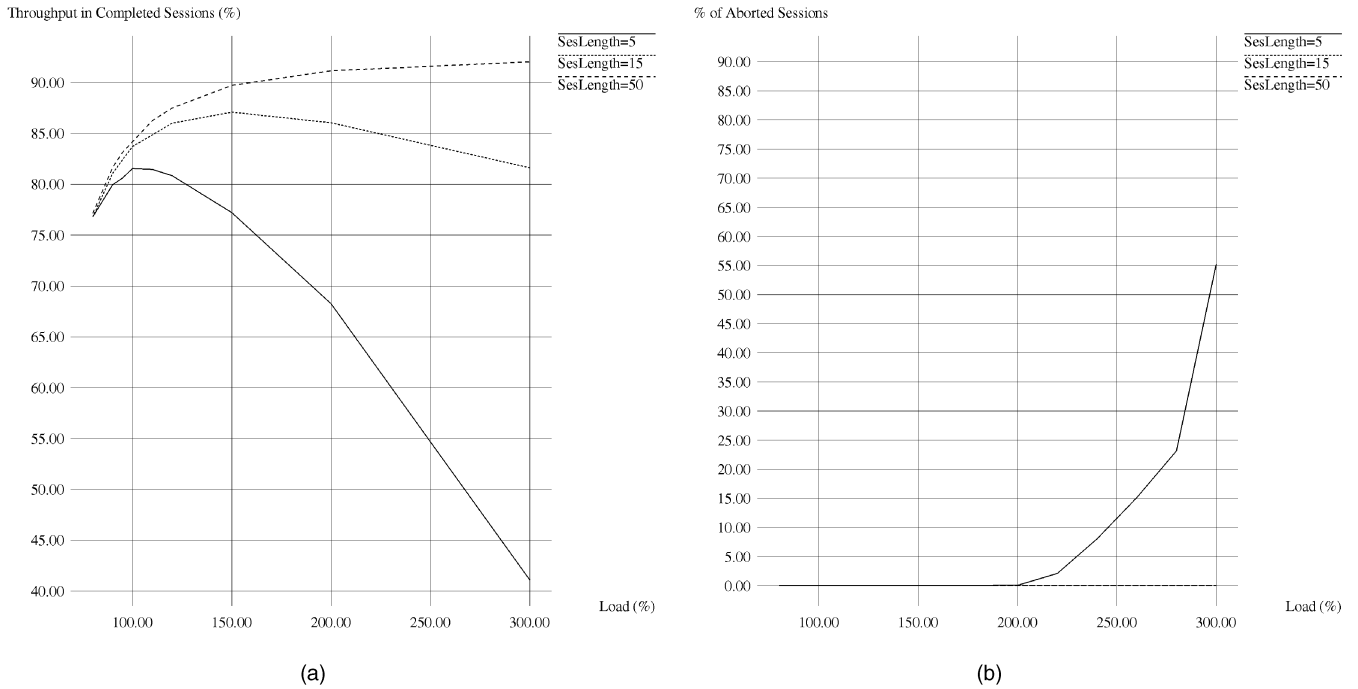

Fig. 6. Web server performance with SBAC: (a) Throughput in completed sessions. (b) Percentage of aborted sessions admitted from those admitted for processing.

mean of 15 and 50 are perfect across the whole load space. They meet the desired quality of service requirement: zero aborted sessions from those accepted for service. For a workload with a mean of 5, the results are getting worse at a load greater than 200 percent, i.e., the server, even with the admission control mechanism, still admits more sessions than it can process. In Section 12, we will discuss the cause of this problem in more detail and propose a solution to resolve this situation.

One of the main goals of the admission control mechanism is to ensure completion of any accepted session, independent of a session length. Fig. 7a shows the average session length of completed sessions against the average session length of all generated sessions as the model input. The results are perfect for sessions with a mean of 15 and 50 across the whole load space. For a workload with a mean of 5, the results are getting slightly worse at load around 300 percent of server capacity. The admission control

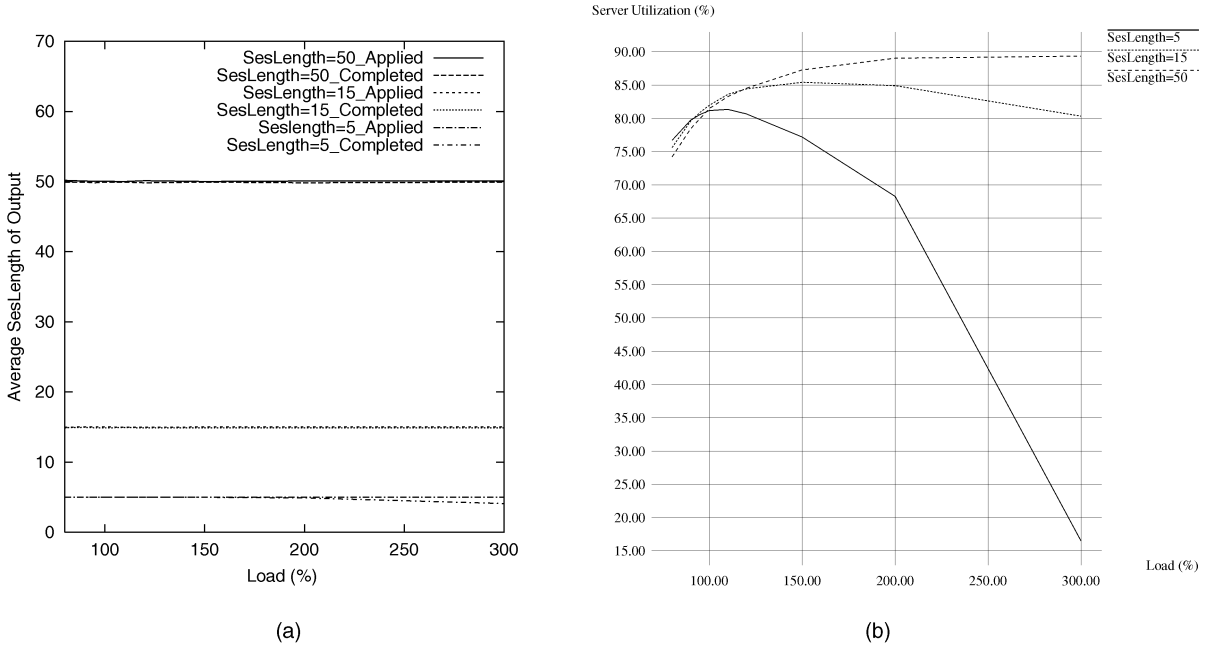

Fig. 7. Overloaded server with admission control: (a) Average length of completed sessions. (b) Useful utilization.

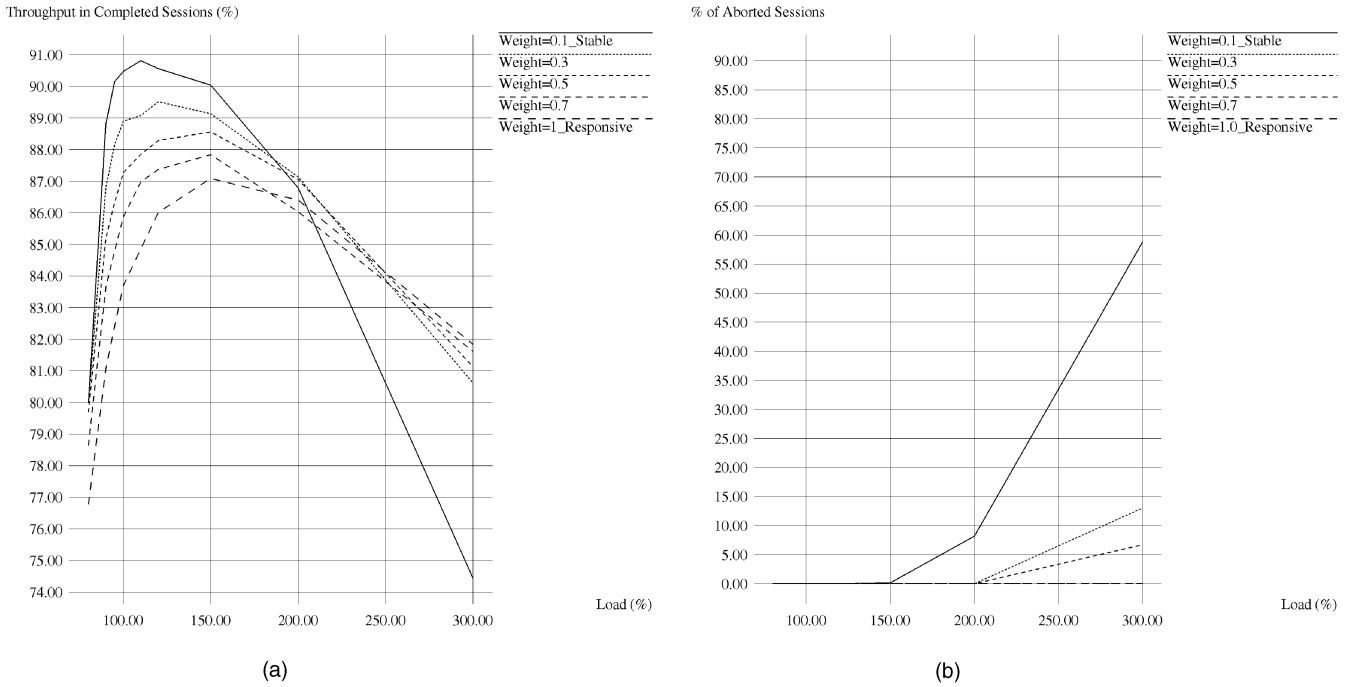

Fig. 8. Performance of SBAC for the family of ac-functions: from ac-stable to ac-responsive and workload with average session length of 15: (a) Throughput in completed sessions. (b) Percentage of aborted sessions.

mechanism dramatically improves the "quality" of the web server output compared with the similar results for the web server with no admission control shown in Fig. 4b.

Finally, Fig. 7b shows useful server utilization as a function of load and session length. Again, for sessions with a mean of 15 and 50, the results are improved by almost an order of magnitude in the overloaded area compared with similar results for the web server with no admission control shown in Fig. 3b. A slight decline for a curve, characterizing a server running sessions with a mean of 15, is due to the rejection messages overhead (it accounts for almost 14 percent of server utilization). Useful server utilization for a workload with a mean of 5 is expectedly lower for 300 percent load due to the increased number of aborted sessions and additional overhead caused by the significant amount of rejection messages sent by the server.

10 Tuning the Admission Control Mechanism for Better QoS

Choosing the right parameters for the admission control mechanism is very important.

Varying an ac-threshold U_{ac} from 95 percent to 97 percent will slightly increase throughput in completed sessions at a price of a greater number of aborted sessions, too, especially for workloads with shorter average session length. Conversely, decreasing an ac-threshold U_{ac} from 95 percent to 93 percent will improve the quality of output, decreasing the number of aborted sessions, but at a price of a slight decrease of throughput in completed sessions.

An ac-weight parameter in the definition of ac-function allowed to define a family of ac-functions has much stronger impact: from ac-stable one to ac-responsive one. Fig. 8a shows the server throughput while running workload with average session length of 15, depending on ac-weight k used in the ac-function f_{ac} definition.

As expected, the server throughput is higher under "more stable" ac-functions for a load below 170 percent. The situation changes for a higher load in favor of "more responsive functions." The rates of aborted sessions are worse for "more stable functions" in a higher load area, as shown in Fig. 8b. This shows again that ac-stable admission control functions achieve better throughput in the load range of 85-120 percent at a price of a higher number of aborted sessions under higher loads. While ac-responsive admission control functions lead to more restrictive admission policies and achieve higher quality of service guarantees, especially at high loads, but at the price of slightly lower server session throughput (in particular, when a server operates at loads in the range 85-120 percent).

Clearly, the desirable goal is to design an adaptive admission control policy, which will dynamically adjust its ac-weight parameter accordingly to vary the load to be more stable, or a more responsive policy to achieve higher quality of service while, at the same time, achieving better server throughput in completed sessions.

11 Hybrid AC-Strategy

Once a web server is augmented with an admission control mechanism, the following question arises: How do we measure the quality and efficiency of this mechanism in practice? The following two values help to reflect admission control "goodness":

- First of all, the percentage of aborted requests which a server can determine is based on the client side closed connections. Aborted requests indicate that

the level of service is unsatisfactory. Typically, aborted requests lead to aborted sessions and could serve as a good warning sign of degrading server performance.

- Second, a percentage of the "connection refused" messages sent by a server, in the case of full listen queue. Refused connections are the dangerous warning sign of an overloaded server and its inevitable poor session performance.

If both of these values are zero, then it reflects an admission control mechanism that uses an adequate ac-function f_{ac} to cope with current workload and traffic rate. Occurrences of aborted requests or refused connections reflect that ac-function f_{ac} has to be more responsive.

From the other side, if the percentage of aborted requests and refused connections is zero, it could also be the case that ac-function f_{ac} is too restrictive and, hence, the server is rejecting some of the sessions which it could handle otherwise. This results in decreased server throughput.

Our goal is to design an admission control strategy which minimizes the percentage of aborted requests and refused connections (ideally to 0) and maximizes the achievable server throughput in completed sessions.

We use the requirements discussed above to design a self-tunable admission control strategy, called *hybrid*.

The idea is very natural: Once some aborted requests or refused connections are observed, the admission strategy is adjusted to be ac-responsive, which is the most restrictive admission policy. After that, this strategy is kept unchanged for a time long enough to observe an "average session life." If, during this time interval, there are no aborted requests or refused connections, then the strategy is adjusted to be "slightly less responsive." In such a way, the admission strategy is to try to migrate closer to the ac-stable strategy until occurrences of aborted requests or refused connections signal the necessity to switch to the ac-responsive strategy. There is some similarity between this idea and the method the internet protocol TCP-IP uses to adjust itself in the presence of congestion.

Let $Ab(i)$ denote the number of aborted requests and refused connections accumulated during the ac-interval T_i. Let *ac-cycle* define the number of time intervals we will observe f_{ac} for aborted requests and refused connections before we adjust the ac-function to be less responsive. In the simulation model, it is defined by $ThinkTime \times SesLength$, which is an approximation of an average session "life." This time interval aims to reflect a cycle (from the sessions admission to their completion) of SBAC working with a new, adjusted ac-function and it is estimated to be long enough to evaluate the "goodness" of this function. We will discuss later, in Section 12, how to approximate the *ac-cycle* in practice.

The *hybrid* strategy adjusts its admission function f_{ac} in the following two situations:

- Let $Ab(i) > 0$ during the time interval T_i. Then, for the next time interval T_{i+1}, the ac-weight k in ac-function f_{ac} is adjusted to 1 (i.e., $k = 1$), changing f_{ac} to become the ac-responsive function with the most restrictive admission policy.
- Let the number of the aborted requests and refused connections stay equal to zero since the last time admission function was adjusted and for the whole duration of the *ac-cycle*:

$$\sum_{j=i+1}^{i+1+ac_cycle} Ab(j) = 0.$$

Then, for the next time interval T_{n+1} (where $n = i + 1 + ac_cycle$), the ac-weight k in ac-function f_{ac} is decreased by 0.1 (i.e., $k = k - 0.1$), changing f_{ac} to become a slightly less "responsive" ac-function with less restrictive admission policy.

The two steps described above repeat, depending on the situation. If, for the next *ac-cycle*, the number of aborted requests and refused connections is zero, then the ac-weight k in ac-function f_{ac} is decreased further by 0.1 ($k = k - 0.1$) and it continues to adjust in a similar way until it becomes an ac-stable policy. Otherwise, if, for some time interval T_j during the *ac-cycle*, $Ab(j) > 0$, then, as was described before, ac-weight k is adjusted to 1 again, changing f_{ac} back to an ac-responsive function. In such a way, f_{ac} adjusts and tunes itself between the ac-stable and ac-responsive policies according to observed traffic load variations.

12 Predictive AC-Strategy

One of the goals of the admission control mechanism is to minimize the number of aborted sessions by explicit session rejection. Fig. 6b shows the percentage of aborted sessions for a server augmented with *SBAC*, using ac-responsive strategy and ac-threshold $U_{ac} = 95\%$ running workloads, with the average session lengths of 5, 15, and 50. The results for sessions with a mean of 15 and 50 are perfect across the whole load space. They meet the desired level of service requirement: zero aborted sessions from those accepted for service.

For a workload with a mean of 5, the results are getting worse at a load greater than 200 percent. At 300 percent, up to 55 percent of admitted for processing sessions are aborted. And, it is happening even when we are using ac-responsive strategy, which provides us with the most restrictive admission policy. The reason is that the shorter the average session length, the higher the number of sessions generated by the clients and accepted by the server during the ac-interval. One way to fix the problem is to reduce the ac-interval. However, it is not always possible to reduce the ac-interval to a desirable value. For some operating systems, cpu utilization is updated on a base of a 5 second interval. Fig. 9 shows the percentage of aborted sessions from those admitted for processing for an admission control mechanism with an ac-interval of 5 seconds. The results for SBAC based on an ac-interval of 5 seconds are unsatisfactory for a whole family of workloads with an average session length less than 50.

A general reason why the *SBAC* mechanism based on CPU utilization measurements can break under certain rates and not work properly is the following: The decision, whether to admit or reject new sessions is made at the

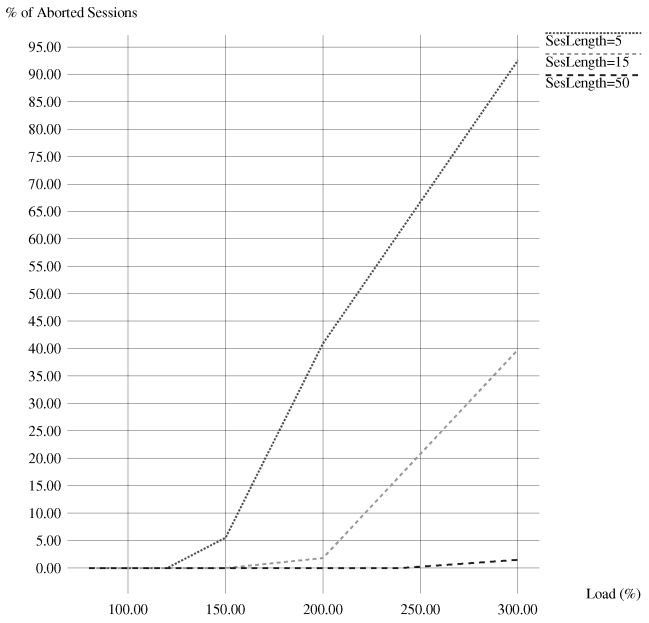

Fig. 9. Percentage of aborted sessions from those admitted for processing for a server with admission control under ac-interval = 5 sec.

boundaries of the ac-intervals. Thus, the model is the "on-off" control model. Once the decision is made to accept the arriving new sessions, i.e., it is in the "on" mode, this decision cannot be changed until the next ac-interval. However, in the presence of a very high load, the number of accepted new sessions may be much greater than server capacity and it inevitably leads to aborted sessions in the future and poor session completion characteristics. The only way to avoid the situation described above is:

- to estimate the number of sessions a server is able to process (note that it depends on the design and content of a particular e-commerce web site, as well as a specific transaction mix defined by the customer behavior pattern typical for this site) and
- to admit, during the time interval, no more sessions than the estimated number prescribes.

To correctly estimate the number of sessions a server is able to process per time interval, we need to take into account the session rejection overhead. For workloads with short and medium average session length, the rejection overhead can be significant for high traffic loads. But, even when it is only 5 to 10 percent, this overhead should be taken into account since a small inaccuracy tends to accumulate over a longer period of time.

In Section 8, we derived an upper bound for the rejection overhead as a function of the applied load and average session length. Once we have estimated the rejection overhead (see (5) in Section 8), it is easy to predict the number of sessions a server is capable of processing per time interval. It is derived by replacing x with (5) in (4) from Section 8:

$$y = \frac{S_r * (SesLength - Load)}{SesLength \times (SesLength - 1)}, \qquad (7)$$

where x is the number of rejected sessions per second and y is the number of completed sessions per second.

Predictive admission control strategy is defined in the following way: For each ac-interval T_i, it predicts the number of sessions a server is able to process, depending on the load and workload characterization. The web server accepts this number of new sessions and rejects any new session above this quota.

Formula (7) depends on three parameters: S_r is the request rate per second a server can process, *Load* is the new sessions arrival rate, and *SesLength* is an average session length.

How can these parameters be obtained in practice?

The request rate per second S_r that a server can process (for this particular workload) is an easily measured parameter. The web server keeps a running counter of accepted sessions C_s (the C_s is incremented for each accepted session by one) and a running counter of requests C_r related to the accepted sessions (the C_r is incremented for each processed request belonging to an accepted session). This provides an approximation of the average session length: $SesLength = \frac{C_r}{C_s}$. After that, using the average session length, the server capacity in sessions S_s can be computed using (1). Finally, by counting the number of new sessions arrivals, *Load* can be evaluated. An approximation of *ac-cycle*, discussed in Section 11, can be done by measuring interrequest time (it is, essentially, the sum of the request response time and the client think time) multiplied by the average session length.

Remark. Clearly, the efficiency of the predictive strategy depends on an accuracy of our prediction. The strategy works much better when one keeps track of possible inaccuracies occurring, for example, as a result of rounding up fractions. A more serious source of inaccuracy can occur because of mispredicting the *Load* since our prediction is based on the previous interval. For example, the *Load* during the previous ac-interval was 200 percent and we estimated, using (7), how many sessions can be accepted during the following time interval. However, later analysis of the *Load* during this time interval shows that it was 300 percent. It leads to some mismatch, easily computed using (7): We accepted slightly more sessions than is allowed since we assumed a slightly smaller rejection overhead (or the situation can be reversed). In order to eliminate further accumulation of such inaccuracy, the next ac-interval quota has to be adjusted (increased or decreased) by the computed sessions amount. In our simulation model, we implemented a predictive strategy which adjusts possible inaccuracy as well as evaluates an amount of unused quota for the last few ac-intervals to allow its usage in the near future.

13 Comparison of AC-Strategies

The new adaptive admission control strategies, hybrid and predictive, are designed to complement shortages of ac-stable and ac-responsive strategies. Since these shortages show up under different load conditions, we designed two variable-load traffic patterns to verify whether the new

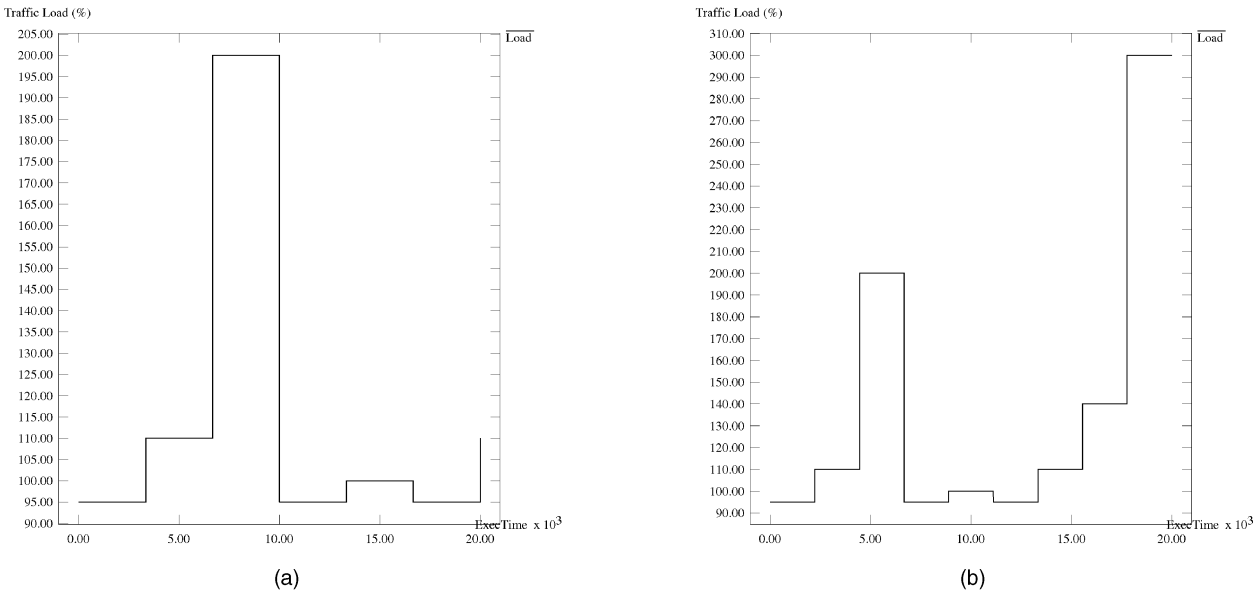

Fig. 10. (a) "Usual day" workload traffic pattern. (b) "Busy day" workload traffic pattern.

admission strategies adequately adjust their behavior depending on traffic load.

First, the traffic pattern is defined by the pattern shown in Fig. 10a. We call it the *"usual day"* traffic pattern. It only has a few intervals of not very high overload and, the rest of the time, it has a load close to the server capacity. This type of load might be typical in practice: Most of the time, the load is manageable, only occasionally exceeding server capacity. The second workload is defined by the pattern shown in Fig. 10b. We call it the *"busy day"* workload. This traffic pattern spends half of the time in overload (reaching a peak of 300 percent during one of the intervals) and, for the other half of the time, it has a load close to a server capacity. We do not include a "bad day" workload (with consistently high overload for all intervals) since the results are predictable and we will comment on them at the end of the section.

Fig. 11a and Table 2a show the results for a *"usual day"* traffic pattern: server throughput in completed sessions and percentage of aborted sessions. These simulation results demonstrate that, for a workload with an average session length of 5, even for a "usual day" traffic load, ac-stable strategy has 13.5 percent of aborted sessions (from accepted ones), while ac-responsive strategy has no aborted sessions, but its throughput is 6 percent less than the throughput of the ac-stable strategy. The hybrid strategy has the same throughput as the ac-stable strategy throughput (even slightly better) and only 1.5 percent of aborted sessions. Thus, the proposed hybrid strategy improves server throughput while supporting a high quality of service-level: a very low number of aborted sessions. The predictive strategy shows even better results. It outperforms all of the strategies: The server throughput is improved by 14 percent comparing with the ac-responsive strategy and has no aborted sessions.

The simulation results for workloads with average session length of 15 and 50 are similar. The rates of aborted sessions are significantly less for all the strategies. All of the strategies are able to provide high levels of service. However, the hybrid and predictive ac-strategies support higher server throughput in completed sessions.

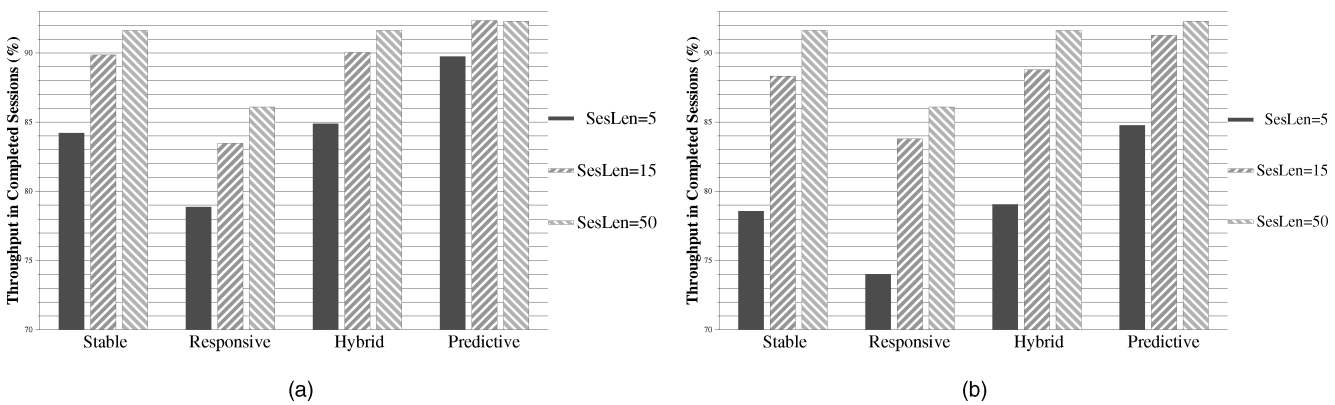

Fig. 11. Throughput in completed sessions: (a) "Usual day" workload. (b) "Busy day" workload.

TABLE 2
Percentage of Aborted Sessions: (a) "Usual Day" Workload, (b) "Busy Day" Workload

Percentage of Aborted Sessions: "Usual Day" Workload)			
AC Strategy	Session Length		
	5	15	50
Stable	13.47%	0.88%	0%
Responsive	0.12%	0%	0%
Hybrid	1.63%	0.07%	0%
Predictive	0.12%	0%	0%

(a)

Percentage of Aborted Sessions: "Busy Day" Workload)			
AC Strategy	Session Length		
	5	15	50
Stable	27.1%	12.92%	0%
Responsive	13.04%	0.55%	0%
Hybrid	13.25%	1.67%	0%
Predictive	0.27%	0.15%	0%

(b)

Fig. 11b and Table 2b show the results for a *"busy day"* traffic pattern. For a *"busy day"* traffic pattern, the number of aborted sessions is generally higher, especially for a workload with average session length of 5. The ac-stable strategy has 27 percent of the aborted sessions for this workload. Moreover, even the ac-responsive strategy is unable to provide a satisfactory level of service: It exhibits 13 percent of the aborted sessions. Since the hybrid strategy is a special combination of the ac-stable and ac-responsive strategies, it also has 13 percent of the aborted sessions with, however, higher sessions throughput. Thus, none of these three strategies provides an acceptable level of service for workloads with short average session length. The predictive strategy produces the best results. It provides both the best overall server throughput (14 percent improvement) while having no aborted sessions (or almost no aborted sessions: 0.27 percent).

Simulation results for workloads with an average session length of 15 and 50 are similar. The rates of the aborted sessions are less for all the strategies in this study. Only the ac-stable strategy fails to provide an adequate level of service: It still has 13 percent of the aborted sessions. However, the hybrid strategy improves the situation: It has only 1.6 percent of the aborted sessions and a 6 percent improvement in throughput. The predictive strategy again provides the best overall results.

For a "bad day" traffic pattern (with consistently high overload for all intervals), the results are predictable and show the same tendency observed for a "busy day" traffic pattern. For workloads with short average session length, the only strategy which works consistently well is the predictive one. For workloads with medium and long average session length, both the hybrid and predictive strategies provide the best results.

14 Conclusion

Today's web servers often perform poorly under overload. In order to meet quality of service requirements, system administrators are forced to significantly overprovision their sites. In this paper, we introduced a new, session-based workload for measuring web server performance. We showed that an overloaded web server can experience a significant loss of throughput in the number of completed sessions compared against the server throughput measured in requests per second. However, this loss is not always easy to recognize. When the session lengths are exponentially distributed (in other words, there is enough variability in session lengths), the throughput in sessions for an overloaded server decreases slightly, but not dramatically.

Analysis of the completed sessions reveals, however, that the majority (up to 98 percent) of completed sessions are short: The overloaded web server discriminates against the long sessions. This could significantly impact sales and profitability of commercial web sites because the sale-sessions are typically 2-3 times longer than nonsale ones. Based on this analysis, we formulate the web quality of service requirements that a web server has to support. In particular, a fair guarantee of completion, for any accepted session, independent of a session length, is a crucial requirement for a commercial web site to be successful.

Additionally, we proposed two adaptive, self-tunable admission control strategies, hybrid and predictive ones, aimed at optimizing the performance of the *SBAC* mechanism and at improving the quality of service provided by the SBAC. These strategies account for the bursty nature of web traffic and they adjust and recompute the ac-mechanism parameters accordingly. The proposed hybrid and predictive admission policies allow the design of a powerful admission control mechanism which tunes and adjusts itself for better performance across different workload types and different traffic loads.

We showed that a web server augmented with an admission control mechanism is able to provide the required web quality of service guarantees. Incorporating this technique into a product allows HP to offer solutions to customers that enables them to migrate core business functions onto web-based technologies and to use web applications for strategic advantage.

Acknowledgments

The authors would like to thank Sky Golightly for his help in obtaining web server log files from e-commerce web sites. They would also like to thank Scott Jorgenson for his work in developing a productive version of session-based admission control and for providing data that helped validate the technology. Their special thanks go to the anonymous referees for numerous suggestions, useful remarks, and production discussion which helped to improve the content and presentation of this paper. This

work was performed while Peter Phaal was working at HPLabs.

REFERENCES

[1] J. Almeida, M. Dabu, A. Manikutty, and P. Cao, "Providing Differentiated Levels of Service in Web Content Hosting," *Proc. First Workshop Internet Server Performance,* June 1998.

[2] M. Arlitt, "Characterizing Web User Sessions," HP Labs Report No. HPL-2000-43, 2000.

[3] T. Abdelzaher and N. Bhatti, "Web Content Adaptation to Improve Server Overload Behaviour," *Proc. Eighth Int'l World Wide Web Conf.,* May 1999.

[4] C. Aurrecoechea, A. Campbell, and L. Hauw, "A Survey of QoS Architectures," *ACM/Springer-Verlag Multimedia Systems J.,* special issue on QoS architecture, vol. 6, no. 3, May 1998.

[5] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proc. 1998 ACM Sigmetrics,* 1998.

[6] N. Bhatti and R. Friedrich, "Web Server Support for Tiered Services," *IEEE Network J.,* vol. 13, no. 5, Sept./Oct. 1999.

[7] P. Bhoj, S. Ramanathan, and S. Singhal, "Web2K: Bringing QoS to Web Servers," HPLabs Report No. HPL-2000-61, May 2000.

[8] L. Cherkasova and P. Phaal, "Session Based Admission Control: A Mechanism for Improving Performance of Commercial Web Sites," *Proc. Seventh Int'l Workshop Quality of Service,* 31 May-4 June 1999.

[9] A. Cohen, S. Rangarajan, and H. Slye, "On the Performance of TCP Splicing for URL-Aware Redirection," *Proc. Second Usenix Symp. Internet Technologies and Systems,* 1999.

[10] M. Colajanni, P. Yu, V. Cardellini, M. Papazoglou, M. Takizawa, B. Cramer, and S. Chanson, "Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers," *Proc. 18th Int'l Conf. Distributed Computing Systems,* May 1998.

[11] R. Enhelscall, "Balancing Your Web Site. Practical Approaches for Distributing HTTP Traffic," *WEB-Techniques,* vol. 3, no. 5, 1998.

[12] S. Jamin, P. Danzig, S. Shenker, and L. Zhang, "A Measurement-Based Admission Control Algorithm for Integrated Services Packet Networks," *Proc. SIGCOMM Symp. Comm. Architectures and Protocols,* 1995.

[13] V. Kanodia and E. Knightly, "Multi-Class Latency-Bounded Web Services," *Proc. Eighth Int'l Workshop Quality of Service,* 2000.

[14] K. Li and S. Jamin, "A Measurement-Based Admission Controlled Web Server" *Proc. IEEE INFOCOM 2000,* Mar. 2000.

[15] D. Menansce, V. Almeida, R. Fonseca, and M. Mendes, "A Methodology for Workload Characterization of e-Commerce Sites," *Proc. ACM Conf. Electronic Commerce,* 1999.

[16] D. Menansce, V. Almeida, R. Fonseca, and M. Mendes, "Resource Management Policies for e-Commerce Servers.," *Proc. Second Workshop Internet Server Performance,* June 1999.

[17] D. Mosberger and T. Jin, "Httperf—a Tool for Measuring Web Server Performance," *Performance Evaluation Rev.,* vol. 26, no. 3, Dec. 1998.

[18] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Drushel, W. Zwaenepoel, and E. Nahum, "Locality-Aware Request Distribution in Cluster-Based Network Servers," *Proc. Eighth Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* 1998.

[19] R. Pandey, J. Barnes, and R. Olsson, "Supporting Quality of Service in HTTP Servers," *Proc. 17th Ann. SIGACT-SIGOPS Symp. Principles of Distributed Computing,* 1998.

[20] H. Schwetman, "Object-Oriented Simulation Modeling with C++/CSIM," *Proc. 1995 Winter Simulation Conf.,* 1995.

[21] "The Workload for the SPECweb96 Benchmark," http://www.specbench.org/osg/web96/workload.html, year?

[22] W. Tang, L. Cherkasova, L. Russell, and M. Mutka, "Modular TCP Handoff Design in STREAMS-Based TCP/IP Implementation," *Proc. First Int'l Conf. Networking (ICN-2001),* 2001.

Ludmila Cherkasova received the PhD degree in computer science from the Novosibirsk Institute of Computing Systems, Russia, in 1984. At that time, she did research in concurrency theory and its application to distributed systems. She was an invited speaker at the International Symposium on Mathematical Foundation of Computer Science (MFCS '88) and coauthored several papers in leading theoretical conferences and journals (MFCS, STACS, FCT, PNPM, Acta Informatica). She joined Hewlett-Packard Laboratories, Palo Alto, California, in 1991, where she is currently a senior scientist in the Internet and Computing Platforms Technologies Center at HPLabs, Palo Alto. Her current research interests are in computer systems, network protocols, and characterization of next generation system workloads, with a focus on emerging applications in the internet data center infrastructure. She is the industrial chair of IEEE WECWIS 2002 and program committee cochair of the Workshop on Web Engineering, Italy, 2002.

Peter Phaal received an electrical engineering degree from the University of the Witwatersrand, Johannesburg, South Africa, in 1994 and, in 1996, the PhD degree in control engineering from Cambridge University, Cambridge, England. He worked as a member of the technical staff at Hewlett-Packard Laboratories, Bristol, England, where he conducted research into network management, developing statistical sampling technologies for monitoring network traffic. Later, he moved to Hewlett-Packard Laboratories, Palo Alto, California, where he researched the performance of Internet-based services, developing admission control mechanisms that improve the overload performance of web servers. He is author of the book *LAN Traffic Management* (Prentice Hall, 1994) and has been granted more than 10 patents in the field of network management. In 1999, he founded InMon Corporation, a San Francisco based company developing network management software and instrumentation for monitoring traffic in high-speed, switched networks. He is currently president of InMon Corp.

▷ **For more information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.