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.

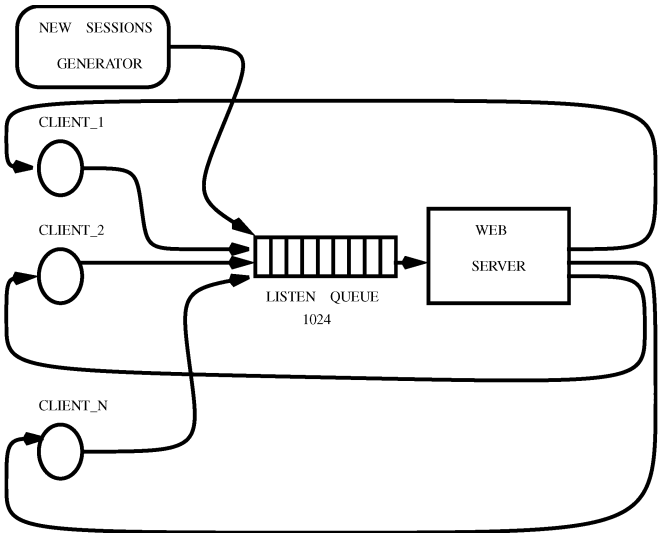


Fig. 1. The basic structure of the simulation model.

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

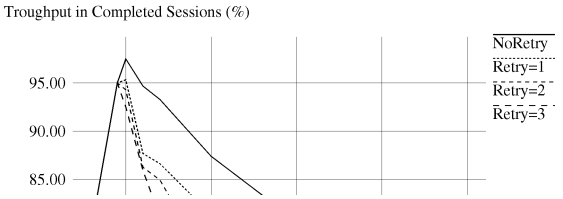
Since we would like to evaluate a web server performance in terms of successfully completed sessions—the session length distribution is one of the main parameters in our sensitivity study of overloaded web server performance. As for the other client parameters such as timeout, the number of retries,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

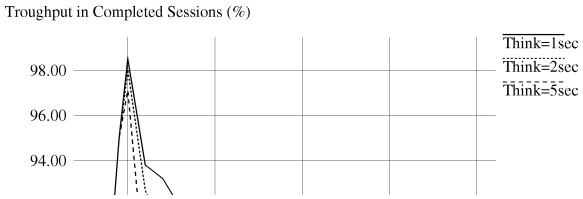
There is a simple relation based on

- 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.





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.