Predictive Admission Control Strategy for Overloaded Commercial Web Server

Ludmila Cherkasova
Hewlett-Packard Laboratories
1501 Page Mill Road,
Palo Alto, CA 94303
cherkasova@hpl.hp.com

Peter Phaal*
InMon Corp.
1404 Irving St.
San Francisco, CA 94122, USA
Peter_Phaal@inmon.com

Abstract *We use a session-based workload to measure a web server's performance. An overloaded web server can experience a severe loss of throughput when measured as the number of completed sessions. Moreover, the overloaded web server discriminates against longer sessions. Session based admission control (SBAC), introduced in [2], prevents a web server from becoming overloaded and ensures that longer sessions can be completed. In this paper, we propose a new "predictive" strategy as a robust and flexible admission control mechanism, aiming to optimize the SBAC performance.*

1 Introduction

Internet based e-commerce services continue to grow at rapid pace. These commercial applications impose a set of additional, service level expectations. Customer's visit to e-commerce site consists of a sequence of related requests. Typically, access to a web service occurs in the form of a *session* consisting of many individual requests. Customer of online bookstore, before placing an order through the web site, typically performs the following steps:

- searching the titles of interest;
- browsing the information about selected books such as annotation, contents, reviews, etc.;
- adding chosen books to a shopping cart;
- providing shipping information and the address;
- arranging payment agreement (providing credit card information); and
- finally, receiving a confirmation.

*This work has been done while Peter Phaal was working at HPLabs.

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.

The main goal of a *session based admission control (SBAC)*, introduced in [2], is the prevention of web server overload. 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). In [1, 2], we introduce a simple implementation of session based admission control based on server CPU utilization. The idea of introducing admission control for a web server running a session-based workload to guarantee "WebQoS" closely resembles with admittance of a new flow into resource constrained network in traditional QoS setup [4, 5].

In this paper, we examine a trade off between two desirable properties for an admission control mechanism: *responsiveness* and *stability* and introduce a family of admission control policies (ac-policies) which cover the space between ac-stable and ac-responsive policies. If a server's load during previous time intervals is consistently high, and exceeds its capacity, then responsiveness is very important: the admission control policy 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, then the stability, which takes into account some load history, is a desirable property. It helps to maximize server throughput and does not unnecessary reject newly arriving traffic. The ac-responsive admission control function leads to more restrictive admission policy and achieve better level of service, especially at high loads but at the price of lower server session throughput.

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 strategy consistently shows the best performance results for different workloads and different traffic patterns. For workloads with short average session length, *predictive* strategy is the only strategy which provides both: highest server throughput in completed sessions and no (or, practically no) aborted sessions. Proposed *predictive* admission policy allows 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 Workload Model: Requests and Sessions

To demonstrate the admission control mechanism, we need a workload which have a reasonably high variance of responses, as do today's usage patterns. Most e-commerce sites generate dynamic CGI-created responses.

SpecWeb96 benchmark [7] is an industry standard for measuring Web Servers performance. It is based on generating static HTTP requests to retrieve different length files accordingly to a particular distribution.

SpecWeb96 file mix is defined by the files (requests) distribution from the following four classes:
*0 Class:*100bytes - 900bytes (35%),
1 Class: 1Kbytes - 9Kbytes (50%),
2 Class: 10Kbytes - 90Kbytes (14%),
3 Class: 100Kbytes - 900Kbytes (1%).
This mix provides high variance of responses. For the purpose of this paper, the workload can be thought as a set of CGI-requests with execution time proportional to a "requested SpecWeb96 files" processing time.

In commercial application, a typical access to a web service consists of a sequence of steps (a sequence of individual requests). A transaction is successful only when the whole sequence of requests is completed. The real measure of server performance is the server's ability to process the entire sequence of requests needed to complete a transaction.

We introduce a notion of a session as a unit of session workload. *Session* is a sequence of clients individual requests. The client issues the next request only when it receives a reply for the previous request. The client issues its next request with some time delay, called *think time*. 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 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. For the rest of the paper, we assume the session lengths to be exponentially distributed with a given mean. In order to analyze the server behavior depending on a session length, we have performed experiments for session lengths with a mean of 5, 15 and 50.

3 Server Model: Functionality and Basic Parameters

To compare different admission control strategies, we built a simulation model using C++Sim [6]. Basic structure of the model is outlined in Figure 1. It consists of the following components: 1) a session workload generator; 2) N clients without loss of generality, (we had limited N to 10,000 clients); 3) a web server.

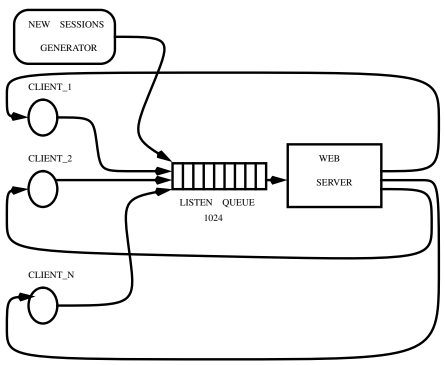

Figure 1: Basic Structure of Simulation Model.

A session workload generator produces a new session request accordingly to specified input model parameters: 1) session load and 2) sessions length distribution. A session request (i.e 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 1024 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 (feed back) loop model: the client issues the next session request only when it receives a reply from the previous request.

Two reasons could cause a request, and a 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 is limited number of *retries*. If the reply still has not been received in time, both the request and a whole session are aborted.

REMARK: When the client receives "connection refused" message due to a full listen queue, it can try to resend the request again. In case of 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.

Without loss of generality, for the rest of the paper, we assume a model with the following client parameters:
1) a think time between the requests of the same session is exponentially distributed with a mean of 5 seconds;
2) a timeout - the time client waits for a reply before resending the request - is set to 1 second (which is a reasonable time, since we do not include network delay in the model);
3) a number of retries to resend the request after timeout is 1.
4) a server capacity is 1000 connections per second for SpecWeb96 file mix.
5) the service time for a file (request) is proportional to the requested file size.

4 SBAC Mechanism and Its Properties

The basic idea of original (measurement based) admission control 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 resent past, it computes an "predicted" utilization. If the predicted utilization gets above a specified threshold then for the next time interval (i.e. the next second), the admission controller will reject all the new sessions and will only serve the requests from already admitted sessions. Once the predicted utilization drops below the given threshold, the server (controller) changes its policy for the next time interval and begins to admit and process new sessions again.

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;

- $T_1, T_2, ..., T_i, ...$ – 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 will distinguish two different values for server utilization:

- $U_i^{measured}$ – a measured server utilization during T_i – the i-th 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 will 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 *ac-weight coefficient*.

A web server with an admission control mechanism re-evaluates its admission strategy on intervals $T_1, T_2, ..., T_i, ...$ 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 arrived during T_{i+1} will be rejected, and web server will process only requests belonging to already accepted sessions.

- If $U_{i+1}^{predicted} \leq U_{ac}$ then 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, that is similar to issue an explicit rejection message.

There are two desirable properties for an admission control mechanism: *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 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, then the "slow reaction", stable admission control policy, which takes into account some load history, is a desirable property. It helps to maximize server throughput and does not unnecessary reject newly arriving traffic.

As we can see, these two properties are somewhat contradictory:

- *responsiveness* leads to a more restrictive admission policy. It aims to minimize a number of aborted sessions and to achieve higher levels of service at a price of slightly lower server session throughput (in particular, when a server is heavy loaded but is not yet overloaded).

- *stability* takes into account a server's load history. It allows to delay a 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, and as a result to poorer session completion characteristics.

The value of coefficient k in definition of f_{ac} introduces a family of admission control policies which cover the space between ac-stable and ac-responsive policies. If $k = 1$ then the admission control policy is based entirely on a 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. Let us call this strategy *ac-stable*.

Due to space limitation, we omit the graphs with simulation results for this family of ac-functions. The ac-stable admission control functions achieve better throughput in the load range of 85%-120% at a price of higher number of aborted sessions under higher loads. While ac-responsive admission control functions lead to more restrictive admission policies and achieve better levels of service, 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%).

5 Requirements for "Ideal" Admission Control Strategy: Web QoS Metrics

Once a web server is augmented with an admission control mechanism, the following question arises: how to measure the "goodness" and efficiency of this mechanism in practice?

The following two values help to reflect an admission control "goodness":

- first of all, the percentage of aborted requests, which server can determine 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 "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 that an admission control mechanism 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 percentage of aborted requests and refused connections is zero, it could be also the case, that ac-function f_{ac} is too restrictive and hence, server is rejecting some of the sessions which it can handle otherwise. It results in decreased server throughput.

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

A general reason why a CPU utilization based implementation of $SBAC$ 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 boundaries of ac-intervals. And this decision can not be changed until the next ac-interval. However, in presence of a very high load, the number of accepted new sessions may be much greater than a server capacity, and it inevitably leads to aborted sessions 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 (it depends on a design and a content of a particular e-commerce web site, as well as a typical customer behavior pattern for this site), and
- to admit during the time interval no more sessions, than the estimated number prescribes.

6 Predictive Admission Control Strategy

In order to correctly estimate the number of sessions a server is able to process per time interval, we also need to evaluate 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 5% to 10% only, this overhead should be taken into account, since a small inaccuracy tends to accumulate over longer period of time.

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. number of connections (requests) per second a server can sustain.
- S_s - a server capacity in sessions, i.e. 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% of server capacity).
- x - a number of rejected sessions per second.
- y - a number of completed sessions per second.

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

$$S_s = \frac{S_r}{SesLength} \qquad (1)$$

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

$$x + y = Load * S_s \qquad (2)$$

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

$$y * SesLength + x = S_r \qquad (3)$$

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

$$y = \frac{Load * S_r}{SesLength} - x \qquad (4)$$

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

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

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

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

The rejection overhead 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%. Note, that formula (6) holds for the $Load$ and $SesLength$ values, satisfying the following condition: $Load - 1 \leq SesLength - 1$. For other values, formula (6) is meaningless, and reflects the situation when applied load is above the server's capacity to send even all the rejection messages. Once we have estimated rejection overhead, it is easy to predict the number of sessions a server is capable to process per time interval. It is derived by replacing x with (5) in equation (4):

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

Predictive admission control strategy works in the following way. For each ac-interval T_i it predicts the number of sessions a server is able to process, and the web server accepts this quota, and reject any new session above this quota. Formula (7) depends on three parameters: S_r -request rate per second a server can process, $Load$ - new sessions arrival rate, and $SesLength$ - an average session length.

How these parameters can be obtained in practice?

A maximum request rate S_r which a server can sustain under particular workload is easily measured parameter.

A running counter of accepted sessions C_s (C_s increments for each accepted session by one) and a running counter of requests C_r related to the accepted sessions (C_r increments for each processed request belonging to an accepted session), allows an approximation of the average session length to be computed $SesLength = \frac{C_r}{C_s}$.

Once, the average session length is evaluated, the S_s - a server capacity in sessions, can be computed using (1).

And after that, by counting the number of new sessions arrivals, $Load$ can be evaluated.

7 Comparison of Admission Control Strategies

In this section, we analyze and compare different admission control strategies: *ac-stable*, *ac-responsive*, and *predictive* one.

New *predictive* strategy is designed to complement shortages of *ac-stable* and *ac-responsive* strategies. Since these shortages show up under different load conditions, we designed two variable traffic patterns to verify whether new admission strategy adequately adjusts its behavior depending on traffic rates.

First traffic pattern is defined by the pattern showed in Figure 2). We call it the *"usual day"* traffic pattern. It has only a few intervals of not very high overload, and the rest of the time, it is 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.

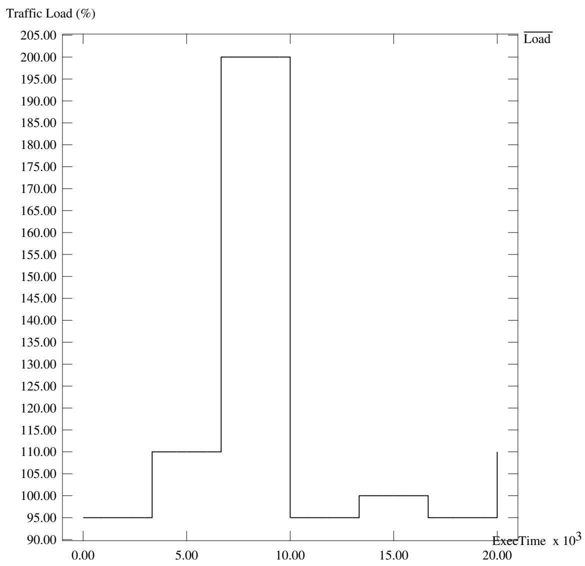

Figure 2: "Usual Day" Workload Traffic Pattern

Second workload is defined by the pattern shown in Figure 3). We call it the *"busy day"* workload. This traffic pattern spends a half of the time in overload (reaching a peak of 300% during one of the intervals), and for the other half of the time it has a load close to a server capacity.

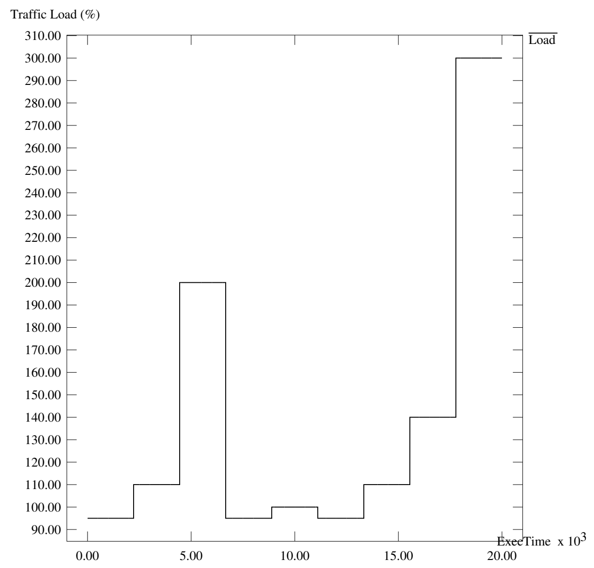

Figure 3: "Busy Day" Workload Traffic Pattern

Figure 4 and Table 1 show the results for a *"usual day"* traffic pattern: server throughput in completed sessions and percentage of aborted sessions.

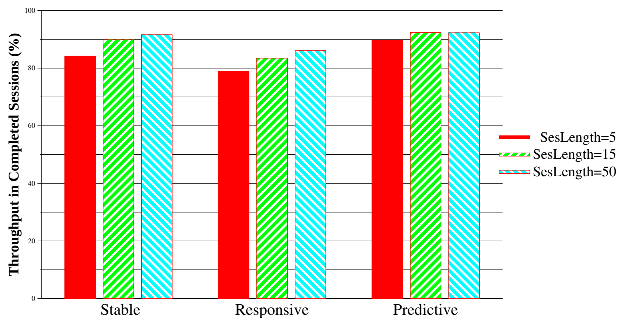

Figure 4: "Usual Day" Workload: Throughput in Completed Sessions.

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%
Predictive	0.12%	0%	0%

(1)

As it was expected, for a workload with average session length of 5, even for "usual day" traffic load, ac-stable strategy has 13.5% of aborted sessions (from accepted ones), while ac-responsive strategy has no aborted sessions, but its throughput is 6% less than throughput of ac-stable strategy. The *predictive* strategy outperforms both ac-stable and ac-responsive strategies: it improves server throughput by 14% comparing with ac-responsive strategy and has no aborted sessions.

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 *predictive* strategy support higher server throughput in completed sessions.

Figure 5 and Table 2 show the results for a *"busy day"* traffic pattern: server throughput in completed sessions and percentage of aborted sessions.

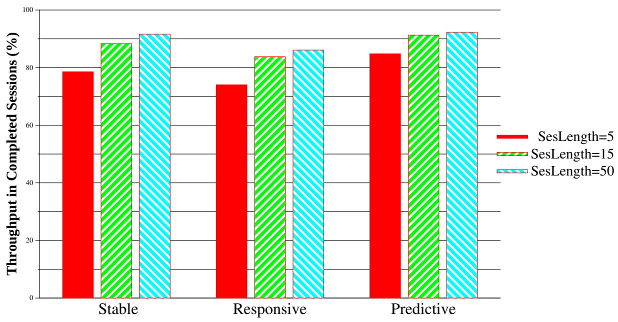

Figure 5: "Busy Day" Workload: Throughput in Completed Sessions.

Percentage of Aborted Sessions: "Busy Day" Workload			
AC Strategy	Session Length		
	5	15	50
Stable	27.1%7	12.92%	0%
Responsive	13.04%	0.55%	0%
Predictive	0.27%	0.15%	0%

(2)

For a *"busy day"* traffic pattern the number of aborted sessions is higher, especially for a workload with average session length of 5. For this workload, ac-stable strategy has 27% of aborted sessions. Even ac-responsive strategy is unable to provide satisfactory level of service: it has 13% of aborted sessions.

The *predictive* strategy produces the best results. It provides both: the best overall server throughput (14% improvement) while having no aborted sessions (or almost no aborted sessions: 0.2%).

Simulation results for workloads with an average session length of 15 and 50 are similar. The rates of aborted sessions are less for all the strategies. Only ac-stable strategy fails to provide adequate level of service: it still has 13% of aborted sessions.

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 *predictive* one.

8 Conclusion

Today's web servers often perform poorly under overload. In order to meet quality of service requirements, system administrators are forced significantly overprovision their sites.

A web server augmented with admission control mechanism is able to provide required web-quality of service guaranties. This technique offer a solution to customers that enables them to migrate core business functions onto web based technologies, and to use web applications for strategic advantage.

In this paper, we propose a new *predictive* admission control strategy, aiming to optimize the performance of *SBAC* mechanism and improve the level of service (number of aborted sessions) provided by *SBAC*.

The *predictive* strategy evaluates the number of new sessions that can be accepted while still guaranteeing that all future session requests will be processed. It consistently shows the best performance result for different workloads and traffic patterns. It improves server throughput (up to 15%) while maintaining a high level of service (less than 0.2% of aborted sessions).

A new, *predictive* admission control strategy outperforms the *"hybrid"* strategy introduced in [3] which is based on a self-tunable admission control function adjusting itself according to variations in traffic loads between the ac-responsive and ac-stable policies. The *hybrid* strategy successfully combines the most attractive features of both ac-responsive and ac-stable policies. Compared to ac-responsive and ac-stable strategies, it shows improved throughput (up to 10%) and high level of service (less than 1% of aborted sessions) for workloads with medium to long average session length. However, for workloads with short average session length, the only strategy, which works consistently well, is predictive one.

Proposed *predictive* admission policy allows to design a powerful admission control mechanism which tunes and adjusts itself for better performance across different workload types and different traffic loads.

References

[1] L. Cherkasova, P. Phaal: ession Based Admission Control: a Mechanism for Improving the Performance of an Overloaded Web Server. HP Laboratories Report No. HPL-98-119, June, 1998. http://www.hpl.hp.com/techreports/98/HPL-98-119.htm

[2] Cherkasova, L., Phaal, P. Session Based Admission Control: a Mechanism for Improving Performance of Commercial Web Sites. In Proceedings of Seventh International Workshop on Quality of Service, IEEE/IFIP event, London, May 31-June 4, 1999.

[3] L. Cherkasova, P. Phaal: Hybrid and Predictive Admission Strategies to Improve the Performance of an Overloaded Web Server. HP Laboratories Report No. HPL-98-125R1, 1998. http://www.hpl.hp.com/techreports/98/HPL-98-125R1.html.

[4] S. Jamin, P. Danzig, S. Shenker, L. Zhang: A measurement-based admission control algorithm for integrated services packet networks. SIGCOMM Symposium on Communications Architectures and Protocols, pp.2-13,1995.

[5] S. Shenker: Is admission control necessary? Infocom'95 Panel " Reservations or No Reservations". ftp://ftp.parc.xerox.com/pub/netresearch/infocom95.html.

[6] Schwetman, H: Object-oriented simulation modeling with C++/CSIM. In Proceedings of 1995 Winter Simulation Conference, Washington, D.C., pp.529-533, 1995.

[7] The Workload for the SPECweb96 Benchmark. http://www.specbench.org/osg/web96/workload.html